ulm university universität
uulm

Modellbasierte Entwicklung von Interaktionsanwendungen

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat. der
Fakultät für Ingenieurwissenschaften und Informatik der
Universität Ulm

vorgelegt von: Guido Manuel de Melo aus
Stuttgart-Bad Cannstatt
Institut für Medieninformatik

2010

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über `http://dnb.d-nb.de` abrufbar.

ISBN 978-3-86853-493-1

Amtierender Dekan:	Prof. Dr.-Ing. Michael Weber
Gutachter:	Prof. Dr.-Ing. Michael Weber
Gutachter:	Prof. Dr. Helmuth Partsch
Tag der Promotion:	9. Februar 2010

© Verlag Dr. Hut, München 2010
Sternstr. 18, 80538 München
Tel.: 089/66060798
www.dr.hut-verlag.de

Die Informationen in diesem Buch wurden mit großer Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. Verlag, Autoren und ggf. Übersetzer übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene fehlerhafte Angaben und deren Folgen.

Alle Rechte, auch die des auszugsweisen Nachdrucks, der Vervielfältigung und Verbreitung in besonderen Verfahren wie fotomechanischer Nachdruck, Fotokopie, Mikrokopie, elektronische Datenaufzeichnung einschließlich Speicherung und Übertragung auf weitere Datenträger sowie Übersetzung in andere Sprachen, behält sich der Autor vor.

1. Auflage 2010

Inhaltsverzeichnis

Inhaltsverzeichnis **iii**

Abbildungsverzeichnis **vii**

1 Einleitung **1**
1.1 Problemstellung . . . 3
1.2 Ziel . . . 3
1.3 Aufbau der Arbeit . . . 4

2 Grundlagen **5**
2.1 Interaktion . . . 6
2.1.1 Dialoge . . . 7
2.1.2 Antropozentrischer Interaktionszyklus . . . 8
2.1.3 Systemzentrischer Interaktionszyklus . . . 9
2.1.4 Inhalt, Logik und Form . . . 11
2.2 Modalitäten . . . 12
2.2.1 Sinnesmodalitäten . . . 12
2.2.2 Modalität als Repräsentation von Information . . . 14
2.2.3 Multimodalität . . . 17
2.2.4 CARE-Eigenschaften . . . 18
2.2.5 Wichtige Modalitätskategorien . . . 18
Grafische Benutzerschnittstellen (GUIs) . . . 19
Sprachbedienschnittstellen . . . 20
Tangible User Interfaces (TUIs) . . . 22
Weitere Modalitäten . . . 22
2.3 Modelle . . . 25
2.4 Fallbeispiele . . . 26
2.4.1 Getränkeautomat . . . 27
2.4.2 MP3-Player . . . 28
2.5 Fazit . . . 29

3 Klassifikation bestehender Ansätze **31**

3.1 Modellarten und Mapping Problem . . . 31
3.1.1 Arten von Modellen . . . 32
3.1.2 Mapping Problem . . . 33
3.1.3 Arten von Mappings und Problemen . . . 35
3.2 Ansätze zur Klassifikation . . . 36
3.2.1 Anforderungen . . . 37
3.2.2 Frühe Aufgliederung nach Abowd et al. . . . 40
3.2.3 Cameleon Framework . . . 40
3.2.4 Fazit . . . 43
3.3 Models and Mappings-Klassifikationsframework . . . 43
3.4 Ansätze zur Beschreibung von Dialogen und Oberflächen . . . 46
3.4.1 Grammatiken . . . 46
3.4.2 Ereignissprachen und Produktionssysteme . . . 47
3.4.3 Zustandsdiagramme . . . 49
3.4.4 Fazit . . . 50
3.5 Abstrakte Modelle für Interaktion . . . 52
3.5.1 ConcurTaskTrees . . . 53
3.5.2 USer Interface eXtensible Markup Language (UsiXML) . . . 56
3.5.3 Unified Modeling Language (UML) . . . 58
3.6 Modelle für Kontext, Evaluation und einzelne Modalitäten . . . 63
3.6.1 Beschreibung von Benutzereigenschaften . . . 63
3.6.2 Gerätemodell . . . 64
3.6.3 Umgebungsmodell . . . 64
3.6.4 Modelle zur Evaluation . . . 65
3.6.5 Notationen für einzelne Modalitäten . . . 66
3.7 Bewertung . . . 67

4 Modellieren von Interaktionsanwendungen mit UML 71
4.1 Domänenmodell . . . 72
4.1.1 Klassendiagramm . . . 73
4.1.2 Einsatz . . . 74
4.2 Aufgabenmodell . . . 74
4.2.1 Use-Case-Diagramme . . . 75
4.2.2 Aktivitätsdiagramme . . . 76
4.2.3 Einsatz . . . 77
4.3 Dialogmodell . . . 78
4.3.1 Zustandsdiagramme . . . 78
4.3.2 Einsatz . . . 79
4.4 Ergänzende Diagramme . . . 82
4.5 Präsentationsmodell . . . 83
4.5.1 Mapping zwischen Aufgaben-, Dialog- und Präsentationsmodell . . . 84
4.5.2 Generierung von Benutzerschnittstellen . . . 86
4.5.3 Direktes Mapping zu Modalitäten . . . 88
4.6 Fazit . . . 88

5 Architekturen für multimodale Benutzerschnittstellen 91
5.1 Ansätze für Architekturen 92
5.1.1 User Interface Management Systeme (UIMS) 92
5.1.2 Seeheim-Modell 93
5.1.3 Arch und Arch-Slinky Modell 94
5.1.4 Multimodale Präsentationskomponenten 96
5.1.5 Kombination der Ansätze zum multimodalen Arch (mArch-Modell) 97
5.2 Zusammenspiel der Komponenten 99
5.2.1 Domänenobjekte und Domänenadapter 99
5.2.2 Dialogsteuerung 100
5.2.3 Präsentationskomponente/Mapper 100
5.2.4 Modalitäten 100
5.3 Fission und Fusion 101
5.3.1 Fission 102
5.3.2 Fusion 103
5.4 Fazit 104

6 Realisierung eines Software-Frameworks 105
6.1 Aufbau des User Interface Management System-Frameworks . . . 106
6.1.1 Parsen von UML Diagrammen 107
6.1.2 Initialisierung des Frameworks 110
6.2 Ausführen von UML 112
6.2.1 Hilfsmechanismus Kaktusstack 114
6.2.2 Aktivitätsdiagramme 115
6.2.3 Zustandsdiagramme 118
6.2.4 Anbindung von Klassendiagrammen 120
6.2.5 Anbindung des Präsentationsmodells 122
Grafische Benutzerschnittstelle 123
Sprachbedienschnittstelle 123
Tangible User Interface (TUI) 124
Zusammenspiel der Komponenten 125
6.2.6 Fusion und Fission 126
6.3 Anwendung des Frameworks 127
6.3.1 Regulärer Einsatz 127
6.3.2 Integration neuer Modalitäten 128
6.4 Fazit 130

7 Fazit und Ausblick 131
7.1 Beiträge 131
7.2 Ausblick 133

A UML Begriffe Englisch – Deutsch 135

Literaturverzeichnis 137

Zusammenfassung **155**

Summary **159**

Dank **161**

Abbildungsverzeichnis

2.1 Felder der MMI nach ACM SIGCHI [9]. Das Gebiet der Dissertation ist farbig markiert. 6
2.2 Interaktionszyklus im Interaction Framework nach ACM SIGCHI [45]. 9
2.3 Interaktionszyklus nach D. Norman [131, S. 47]. Links die Ausführungsphase, rechts die Evaluationsphase. 10
2.4 Interaktionszyklus aus Sicht eines technischen Systems. 11
2.5 Zwei GUIs, links eine Text-basierte Oberfläche zur Paketverwaltung, rechts ein Mobiltelefon mit Touchscreen von HTC (`http://www.htc.com`). 20
2.6 Planung mit URP. Links unten das Wind-Werkzeug (Bild aus [191]). . 23
2.7 Landschaftsplanung mit SandScape. Der Benutzer verformt den Sand und sieht zugleich die projezierten Ergebnisse (Bild aus [79]). 23
2.8 Links ein Kaffeeautomat, rechts die Benutzerschnittstelle eines Automaten als Anwendung nur in Software (Bild links von `http://designnation.de`). 27
2.9 Zwei MP3-Player: Links Apple Ipod, rechts Media Player Classic (`http://mpc-hc.sourceforge.net/`). 28

3.1 Modelltransformationen im Cameleon Framework. 41
3.2 Transformationen und entsprechende Elemente am Beispiel des MP3-Players. 42
3.3 Das Models and Mappings-Klassifikationsframework. 44
3.4 Ausschnitt aus dem Zustandsdiagramm einer Uhr nach der ursprünglichen Arbeit Harels von 1987. 50
3.5 Dialogmodell des MP3-Players als Zustandsdiagramm. 51
3.6 Einordnung von Zustandsdiagrammen in das Klassifikationsframework. 52
3.7 Die Aufgaben des MP3-Players mit ConcurTaskTrees. 54
3.8 ConcurTaskTrees im Klassifikationsframework. 56
3.9 UsiXML im Klassifikationsframework. 58
3.10 Links das Klassendiagramm des Getränkeautomaten, rechts das Aktivitätsdiagramm. 60

3.11 Zustandsdiagramm zur Beschreibung des Verhaltens der Aktion *hello*. Es zeigt die Begrüßung mit Geldeinwurf. 61
3.12 Zustandsdiagramm zur Beschreibung des Verhaltens der Aktion *select*. Es zeigt die Auswahl eines Getränks. 62
3.13 UML im Klassifikationsframework. 63

4.1 Zusammenhang zwischen den eingesetzten UML-Diagrammen. 72
4.2 Links das Klassendiagramm des Getränkeautomaten, rechts das des MP3-Players. 73
4.3 Use-Case-Diagramm mit zwei Use-Cases. 75
4.4 Links das Aktivitätsdiagramm des MP3-Players, rechts das des Getränkeautomaten. 77
4.5 Zustandsdiagramm für den MP3-Player. Der Oberzustand enthält drei Regionen mit parallel aktiven Zustandsdiagrammen. 80
4.6 Zustandsdiagramm des Getränkeautomaten mit Begrüßung und Geldeinwurf. 81
4.7 Zustandsdiagramm des Getränkeautomaten für die Auswahl eines Getränks. 82
4.8 Aktivitätsdiagramm für das Wandeln von MP3 zu PCM. 83
4.9 Links das Aktivitätsdiagramm des MP3-Players, rechts das Zustandsdiagramm der Aktion `play-dialog`. 85
4.10 Glade User Interface Designer, rechts im Kontextmenü die Verknüpfung eines Buttons mit einem Event. 89

5.1 Das Seeheim-Modell. 93
5.2 Das Arch Modell nach [12]. 95
5.3 Das Slinky Metamodell nach [12]. 95
5.4 Komponenten des MVC Konzepts. Zwei parallele Views. 96
5.5 PAC-Agenten in hierarchischer Anordnung. Die Agenten beschreiben die Teile eines Fensters. Der oberste Agent steuert das Fenster und besitzt weder Abstraction noch Presentation. 97
5.6 Die Kombination aus Arch-Slinky und multimodalen Präsentationskomponenten bildet das multimodale Arch-Modell (mArch). 98

6.1 Umsetzung des mArch-Frameworks mit beteiligten Klassen. 107
6.2 Sequenzdiagramm zum Verhalten des mArch-Frameworks beim Start einer Anwendung. UIs registrieren sich beim Builder, dieser initiiert anschließend den Interaktionszyklus. 111
6.3 Verhalten des mArch-Frameworks zur Laufzeit. Die beiden Alternativen (*alt*) zeigen oben den Übergang zwischen zwei Aktionen und unten die Verarbeitung einer Benutzereingabe. 113
6.4 Kaktusstack mit möglichen Operationen. 114
6.5 Die eingesetzten Modellelemente für Aktivitätsdiagramme nach [142]. 116
6.6 Benutzte Modellelemente für Zustandsmaschinen. 118

6.7 Links das Klassendiagramm des MP3-Players, rechts das Zustandsdiagramm. 121
6.8 Ableitung der Klassen `SpeechUI`, `GladeUI` und `WiiUI` von der Klasse `CUI`. 122
6.9 GUI des MP3-Players. 123
6.10 Die Wiimote mit ihren räumlichen Eingabemöglichkeiten. 125
6.11 Links das Aktivitätsdiagramm des MP3-Players, rechts das zugehörige Zustandsdiagramm. 126
6.12 Die wichtigsten Klassen des Frameworks. 127
6.13 Ableitung der Klasse `GladeUI` von der Klasse `CUI`. 129

1 Einleitung

> Shouldn't software be designed so that users could run the same calendar program on a palm-sized device, a laptop, and a wall sized display?
>
> *Ben Shneiderman, [178]*

Heutzutage wird zunehmend mehr Informationstechnik in Alltagsgegenstände integriert und immer mehr Menschen verlassen sich auf Dienste und Informationen, die elektronisch bereitgestellt werden. Die Anzahl kleiner tragbarer Geräte und immer größerer Geräte wie Wanddisplays und Gebäudetechnik, mit denen Benutzer interagieren können, wächst.

Durch die Integration von Technik in Gebäude und Räume, sowie die immer kleineren assistiven Geräte, werden Anwendungen im wörtlichen Sinne „ubiquitär" verfügbar. Mit der wachsenden Rechenleistung ist es mittlerweile auch möglich, große Fortschritte im Sinne von Marc Weisers Ansätzen zum *ubiquitous computing* zu machen [204]. Es ist möglich Alltagsgegenstände zur Interaktion mit digitalen Daten zu verwenden. Der Computer als solches rückt immer mehr aus dem Blickfeld.

Die Nutzung der Geräte ändert sich abhängig von der Umgebung. Im Büro mag ein Benutzer eine klassische Desktop-Umgebung bevorzugen, unterwegs ist es dagegen ein kleines, leichtes Gerät mit geringer Rechenleistung und eventuell Spracheingabe [210], und zu Hause ist es eher eine Projektionswand, um Filme anzusehen. Dabei erwartet ein Benutzer, dass er Anwendungen mit den verschiedenen Geräten in der gleichen Art verwenden kann.

Zusätzlich zur Nutzung von immer mehr Geräten gibt es eine Änderung der Art, in der ein Benutzer mit Anwendungen interagiert. Bei kleinen Geräten erfolgt die Interaktion über eine Tastatur mit nur wenigen Tasten, einen Stift oder durch Berühren des Displays. Einige Geräte bieten parallel dazu eine Sprachsteuerung,

die z. B. bei Mobiltelefonen verbreitet ist. Weitere Möglichkeiten der Interaktion bieten sich bei Konsolenspielen, bei denen die Gesten des Benutzers und ganze Körperbewegungen von den Geräten erkannt werden[1] an. Auch *Tangible User Interfaces*[2] (TUIs), Systeme bei denen der Benutzer Alltagsgegenstände manipuliert um dadurch Daten zu verändern, werden zunehmend erforscht.

Benutzer bevorzugen in dieser zunehmenden Vielfalt an Interaktionsmöglichkeiten solche Geräte und Systeme, die sie ihren Präferenzen entsprechend unterstützen. Mischformen sind je nach Umgebung und Anwendungszweck dementsprechend erwünscht. Weiter ist es interessant, Anwendungen nicht auf ein Gerät festzulegen, sondern vorhandene Interaktionsmöglichkeiten von anderen Geräten mitzubenutzen. Systeme, die durch verschiedene Geräte den Zugang zu Informationen oder Diensten ermöglichen und dabei verschiedene Benutzerschnittstellen gleichzeitig bieten, werden *Multiple User Interfaces* (MUIs) genannt. Alle verwendbaren Geräte müssen dabei koordiniert werden [173], da sich die Benutzerschnittstelle sonst unvorhersehbar verhalten würde.

Diese Vielfalt stellt eine große Herausforderung für die Anwendungsentwicklung dar. Bisher wird in den meisten Fällen eine Anwendung lediglich für eine grafische Benutzerschnittstelle entwickelt. Schon seit den 1990ern existieren Arbeiten wie von Mynatt [121], die GUIs auf Sprachschnittstellen abbilden. Neuere Ansätze wie 2006 von Carter versuchen GUIs auf noch mehr Geräte abzubilden [32]. Jede Anwendung kann sich bezüglich der verwendeten Modalität wie Sprache, TUI oder GUI anders verhalten. Die Nutzer sind dadurch allerdings einer größeren kognitiven Last ausgesetzt, da sie die verschiedenen Abläufe und Verhaltensweisen erst erlernen müssen und es zu Umstellungsschwierigkeiten beim Wechsel der Modalität kommen kann. Einfache Abbildungen zwischen GUI, Sprache und anderen Interaktionsarten sind damit zwar möglich, unterstützen aber den Benutzer nicht optimal in seiner Tätigkeit.

Weil für die Erstellung einer Oberfläche einer Anwendung im Mittel 50% des gesamten Entwicklungsaufwands einer Anwendung verbraucht werden, wie Myers 1992 ermittelt hat [120], sind insbesondere Ansätze, die an dieser Stelle Arbeit sparen, auch ökonomisch interessant. Zwar ist die Studie von Myers schon einige Jahre alt und in der Zwischenzeit sind neue Werkzeuge zur Unterstützung von Entwicklern geschaffen worden, dennoch muss von einem relativ hohen Aufwand für die Entwicklung ausgegangen werden, wenn man den Aufwand für die Vielfalt der unterschiedlichen Benutzerschnittstellen einbezieht.

Es existiert ein starker Trend zu neuen Modalitäten, die viele Schwierigkeiten der Bedienung aktueller Anwendungen reduzieren sollen [144]. Man erwartet sich von ihnen eine leichtere Erlernbarkeit und einfachere Benutzung. Zugleich sind jedoch neue Werkzeuge notwendig, um Anwendungen entwickeln zu können, in die verschiedene Modalitäten wie Sprache oder Gestik integriert werden.

[1] Nintendo Wii über Controller (`http://www.nintendo.com/wii`), Microsoft XBox 360 über Kamera (`http://www.xbox.com/en-US/live/projectnatal/`)

[2] Auch als *reality based user interfaces* (RBIs) bekannt.

Myers stellte 2000 fest, dass die bisherigen Techniken zur Mensch-Maschine Interaktion auf Anwendungen für Geräte mit normalen bis großen Displays und einer Maus optimiert sind und dringend eine Weiterentwicklung für ubiquitäre Anwendungen erfolgen sollte. Dies erfordert verbesserte Methoden zur geräteunabhängigen Spezifikation von Benutzerschnittstellen [117].

1.1 Problemstellung

Für viele Modalitäten existieren heute Beschreibungssprachen. Trotzdem ist es nach wie vor sehr aufwendig für eine Anwendung Benutzerschnittstellen zur Verfügung zu stellen. Dies liegt daran, dass neben einer Vielzahl von Plattformen Eigenschaften der Benutzer wie Präferenzen oder Sehschwächen berücksichtigt werden müssen. Außerdem steigt auch der Entwicklungsaufwand mit der Anzahl der zu unterstützenden Modalitäten. Für jede Modalität wird die komplette Benutzerschnittstelle ein weiteres Mal implementiert, im besten Fall angepasst. Gesucht wird daher eine Möglichkeit die Benutzerschnittstelle modalitätsunabhängig, also abstrakter, zu beschreiben. In Folge würde sich der Aufwand für einzelne Modalitäten deutlich reduzieren.

Eine abstrakte Beschreibung bedeutet, ein Modell zu erstellen. Für Anwendungen, Benutzer, Interaktion und verwandte Bereiche existieren eine Vielzahl von Modellen und Beschreibungssprachen. Die verschiedenen Modelle lassen sich miteinander verbinden und Benutzerschnittstellen für einzelne Modalitäten können durch solche Verknüpfungen abgeleitet werden. Das Anwachsen von Verknüpfungen zwischen Modellen stellt allerdings ein Problem dar. Aus Modellen lassen sich andere Modelle ableiten, auch dies stellt eine Art von Verbindung der Modelle dar. Verknüpfungen müssen bei Veränderungen aktualisiert werden. Dies kann schon bei kleinsten Anpassungen eines Modells eine Vielzahl komplexer Änderungen in anderen Modellen nach sich ziehen.

1.2 Ziel

Ziel dieser Arbeit ist es, geeignete abstrakte Modelle für die Interaktion zwischen Benutzern und Anwendungen zu finden, die die Menge an Verknüpfungen zwischen den Modellen möglichst gering hält.

Schon seit Mitte der 1950ern gibt es Sprachen, die zur Beschreibung von Benutzerschnittstellen geeignet sind [90]. Seitdem ist die Anzahl an Beschreibungssprachen kontinuierlich gewachsen. Deshalb werden Repräsentanten der wichtigsten Sprachkategorien bewertet.

Eine ähnliche Entwicklung ist bei Modalitäten zu beobachten. Die überwiegende Mehrheit an Benutzerschnittstellen machen nach wie vor GUIs aus. Im Lauf der Zeit haben sich aber weitere neue Modalitäten wie etwa Sprache oder Tangible

User Interfaces etabliert. Aufgrund der Menge an Mischformen ist es unmöglich geworden, alle Modalitäten detailliert zu betrachten, deshalb werden einige exemplarisch herausgegriffen. Zur Realisierung wird auf bestehende Ansätze zurückgegriffen, sofern dies möglich und angebracht ist.

Der Schwerpunkt dieser Arbeit liegt auf dem Einsatz von Modellen zur Beschreibung von Interaktionsanwendungen. Das Thema der Generierung von Benutzerschnittstellen aus Modellen wird nur angerissen. Stattdessen wird eine Interpretation der Interaktionsmodelle bevorzugt.

Durch diese Arbeit sollen multimodale Benutzerschnittstellen ermöglicht werden, deren Verhalten durch abstrakte Interaktionsmodelle beschrieben wird. Benutzerschnittstellen werden durch die Trennung der Repräsentation ihrer Inhalte (Oberflächen, Sprache, etc.) von der Interaktionslogik einfacher zu erstellen sein.

1.3 Aufbau der Arbeit

Zu Beginn der Arbeit werden in Kapitel 2 „Grundlagen“ die wichtigsten Begriffe dieser Arbeit vorgestellt und definiert. Nach der Behandlung von Interaktion, Modalitäten und Modellen werden zwei Fallbeispiele vorgestellt, die im Lauf der Arbeit zur Illustration eingesetzt werden.

Kapitel 3 „Klassifikation bestehender Ansätze“ stellt das Problem der Verknüpfungen zwischen Modellen, das *Mapping Problem*, vor. In einem eigens entwickelten Framework zur Klassifikation liegt ein Schwerpunkt der Arbeit. Die Klassifikation wird im selben Kapitel auf bestehende Ansätze angewandt, um eine Modellierungssprache zu bestimmen.

Die folgenden Kapitel 4 „Modellieren von Interaktionsanwendungen mit UML“ und 5 „Architekturen für multimodale Benutzerschnittstellen“ bilden den zweiten Schwerpunkt der Arbeit. Kapitel 4 beschreibt das Vorgehen bei der Modellierung von Interaktionsanwendungen. Kapitel 5 erarbeitet eine Architektur, mit der multimodale Benutzerschnittstellen realisiert werden können.

In Kapitel 6 „Realisierung“ wird das entwickelte Framework zur Realisierung der Architektur vorgestellt. Der Einsatz wird an Fallbeispielen erläutert.

Ein Fazit der Arbeit zieht Kapitel 7, das auch die wissenschaftlichen Beiträge sammelt.

Im Anhang findet sich eine Übersicht über die englischen Begriffe der Unified Modeling Language und deren deutschen Äquivalente.

2 Grundlagen

> Die Wahrheit und Einfachheit der Natur sind immer die letzten Grundlagen einer bedeutenden Kunst.
>
> Paul Ernst, Der Weg zur Form

Bevor Verfahren zur Erstellung multimodaler Benutzerschnittstellen vorgestellt werden, erfolgt eine Einordnung in das Feld der Mensch-Maschine Interaktion (MMI), im englischen *Human Computer Interaction* (HCI). In diesem Kapitel werden außerdem die grundlegenden Begriffe der Arbeit bestimmt und das Arbeitsgebiet genauer beschrieben.

> Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.“
> – Curricula for Human Computer Interaction, ACM SIGCHI [9]

Die *Association for Computing Machinery* (ACM) hat in ihrem Curriculum für dieses Feld [9] die Disziplin der Mensch-Maschine Interaktion umfassend definiert. In Abbildung 2.1 sieht man die beteiligten Bereiche. Diese Arbeit befasst sich dabei mit dem Bereich der Ein-/Ausgabe, Dialogtechniken sowie Architekturen für Dialoge. Die Ein-/Ausgabe bezieht sich auf die Schnittstelle zum Menschen. Unter Schnittstelle werden sowohl die Hardware als auch die Arten der Interaktion von Menschen mit Systemen verstanden, wobei Geräte nur als Mittel zum Zweck begriffen werden. Dialogtechniken beschreiben die Arten von Interaktion mit einem System. Es lassen sich verschiedene Sprachen und Verfahren zu einer Beschreibung von Dialogen verwenden. Im folgenden Kapitel 3 „Klassifikation bestehender Ansätze“ wird hierauf genauer eingegangen.

In diesem Kapitel werden zunächst die Begriffe der Interaktion und des Dialogs genauer geklärt. Zuerst werden die Begriffe Interaktion und Interaktionszyklus eingeführt. Der Begriff eines Modells wird eingeführt und die Verwendungen von Modellen genauer erklärt. Auf Architekturen für interaktive Systeme wird in Kapitel 5 „Architekturen für multimodale Benutzerschnittstellen“ detaillierter eingegangen, um eine geeignete Umsetzung zu erstellen. Zum Schluss des Kapitels werden Fallbeispiele vorgestellt, die in der gesamten Arbeit Verwendung finden.

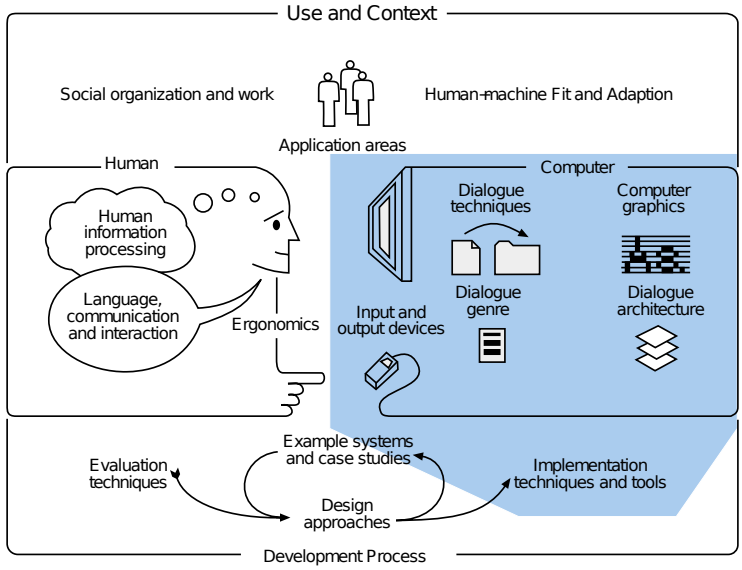


Abbildung 2.1: Felder der MMI nach ACM SIGCHI [9]. Das Gebiet der Dissertation ist farbig markiert.

2.1 Interaktion

Der Begriff Interaktion wird vom Lateinischen *inter* (zwischen) und *agere* (handeln) abgeleitet. In den Sozialwissenschaften kennzeichnet der Begriff die gegenseitige Beeinflussung, die wechselseitige Abhängigkeit und das „Miteinander in Verbindung Treten“ zwischen Individuen und sozialen Gebilden. Im Duden wird Interaktion wie folgt definiert [49]:

Definition 1 (Interaktion) Interaktion, die *(Psychologisch, Soziologisch): aufeinander bezogenes Handeln zweier oder mehrerer Personen, Wechselbeziehung zwischen Handlungspartnern [...].“*

Erweitert man den Begriff ein wenig, so lässt sich darunter auch die Wechselbeziehung zwischen Mensch und Maschine verstehen. Dix et al. nennen jede Kommunikation zwischen Mensch und Maschine Interaktion, unabhängig davon ob diese direkt oder indirekt erfolgt [46, S. 4]. Interaktion kann dabei ein einzelner Austausch oder eine längere Abfolge bezeichnen. Ein *interaktives System* besteht aus mindestens einer Maschine, mit der eine wechselseitige Kommunikation möglich ist.

2.1.1 Dialoge

Die Bedeutung des Begriffs des Dialogs überschneidet sich mit dem der Interaktion. Sobald zwei Partner miteinander kommunizieren sind sie im Dialog. Hierbei steht traditionell die Kommunikation im Vordergrund.

> „*Dialōg* (griech.), Zwiegespräch, gegenseitige mündliche Mitteilung verschiedener, auch einander widerstreitender Ansichten über einen Gegenstand; auch ein Schriftwerk oder Teil desselben in der Form einer solchen Unterredung.“
> – Meyers Konversationslexikon [108]

Definition 2 (Dialog) *Ein Dialog beinhaltet die gesamte Interaktion zwischen den Dialogteilnehmern. Er besteht aus einer logisch zusammenhängenden Menge einzelner Interaktionen.*

Ein Dialog umfasst bei interaktiven Systemen die Gesamtheit der Interaktion, unabhängig von der Art in welcher sie geschieht. Es spielt es nur eine untergeordnete Rolle, ob eine Anweisung an ein System per Sprache oder Knopfdruck übermittelt wird.

Ein interessanter Aspekt dabei ist, dass der Dialogteilnehmer ein Ziel mit dem Dialog verfolgt. Ein Dialog ohne Ziel ist zwar möglich aber bedeutungslos. In der Praxis sind selbst Dialoge wie z. B. das Gespräch über das Wetter von Zielen der Dialogteilnehmer bestimmt [14].

In interaktiven Systemen dienen Dialoge zur Erfüllung bestimmter Aufgaben. Ein Dialog ist z. B. eine Sicherheitsabfrage vor dem Löschen von Dateien oder ein Formular. Ein Überweisungsformular ließe sich beispielsweise in winzige Teildialoge aufteilen, bei denen separat Name, Bankleitzahl usw. abgefragt werden. Dies ist jedoch meist nicht sinnvoll, weshalb in der Definition auch nicht die kleinste Menge von Interaktionen gefordert wird.

Interaktion innerhalb eines Dialogs erfolgt durch Elemente, mit denen ein Benutzer Ein- und Ausgaben durchführen kann.

Definition 3 (Interaktionselement) *Ein Interaktionselement ist ein Element, durch das Interaktion erfolgt. Es kann ein Eingabe- oder ein Ausgabeelement sein.*

Beispiele sind Texteingabefelder, Buttons, Sprachkommandos, und Gegenstände, die in einem *Tangible User Interface* (TUI) benutzt werden.

Schließlich gilt es noch zu definieren wie ein Benutzer mit einem System interagieren kann. Dazu dient die Benutzerschnittstelle. Die Benutzerschnittstelle wird in dieser Arbeit nach der Norm ISO 9241-110 verstanden:

> „Alle Bestandteile eines interaktiven Systems (Software oder Hardware), die Informationen und Steuerelemente zur Verfügung stellen, die für den Benutzer notwendig sind, um eine bestimmte Arbeitsaufgabe mit dem interaktiven System zu erledigen."
> – ISO 9241-110 [77]

Die Steuerelemente von denen die Norm spricht, sind die oben eingeführten Interaktionselemente. Damit erhält man die folgende Definition:

Definition 4 (Benutzerschnittstelle) *Die Benutzerschnittstelle ist die Summe aller Interaktionselemente, die in einem Dialog verwendet werden können.*

2.1.2 Antropozentrischer Interaktionszyklus

Interaktion findet immer in einem Zyklus statt. Ein einfaches Modell setzt sich aus Benutzer, System, Eingabe und Ausgabe zusammen. Die Eingabe ist an das System gerichtet, die Ausgabe an den Benutzer. Eingabe und Ausgabe zusammen ergeben die Schnittstelle. Nachdem sich die Schnittstelle zwischen System und Benutzer befindet, sind vier Schritte für eine Interaktion nötig (vgl. Abbildung 2.2).

Ein Benutzer beginnt den Interaktionszyklus mit einem bestimmten Ziel. Um es zu erreichen nimmt er Eingaben in das System vor, welche dieses intern verarbeitet. Das System erzeugt Ausgaben, an denen der Nutzer die Auswirkungen seiner Eingabe auf das System erkennen kann.

Normans Modell des *execution-evaluation cycle* ist noch deutlich ausführlicher und legt den Schwerpunkt auf die kognitiven Prozesse, die beim Benutzer ablaufen [131]. Die Interaktion wird in zwei sich abwechselnde Phasen zerlegt: *Execution*, die Ausführung einer Aktion, sowie *Evaluation*, die Bewertung der Aktion auf ihren Beitrag zum Erreichen des Ziels.

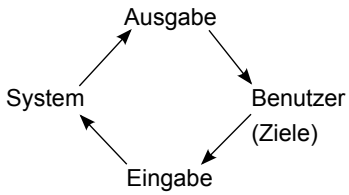


Abbildung 2.2: Interaktionszyklus im Interaction Framework nach ACM SIGCHI [45].

Eine Aktion unterteilt sich in 7 feinere Schritte, die in Abbildung 2.3 zu sehen sind. Der Zyklus beginnt in der Regel beim Benutzer, der ein Ziel erreichen will. In der Ausführungsphase wird dazu die Intention gebildet, etwas im System (im Bild „Die Welt“ genannt) verändern zu wollen. Die Intention wird anschließend in mehrere Aktionen zerlegt, die dann ausgeführt werden und dadurch den Zustand der Welt ändern. Während der Evaluationsphase geschehen drei Dinge. Die Änderungen werden sichtbar und vom Benutzer erfasst. Ihre Bedeutung wird interpretiert und schließlich wird der aktuelle Systemzustand bewertet. Dieser lässt sich dann mit dem Ziel vergleichen und nach Bedarf beginnt der Zyklus mit einem veränderten oder neuen Ziel erneut.

2.1.3 Systemzentrischer Interaktionszyklus

Statt aus Benutzersicht lässt sich der Interaktionszyklus auch aus Sicht des technischen Systems betrachten. Der Zyklus kann hier ebenfalls in mehrere Schritte untergliedert werden. Abbildung 2.4 zeigt die Schritte im gesamten Zyklus aus Sicht eines technischen Systems. Ein System beginnt den Zyklus, indem es Informationen über seinen internen Zustand aufbereitet und einem Benutzer mitteilt. Dieser tätigt seine Eingabe, die den Zustand des Systems ändert und der Zyklus beginnt von neuem. Der Interaktionszyklus kann generell in physikalische und logische Interaktion unterteilt werden[1].

Mit logischer Interaktion bezeichnet man die Interaktion, die ein Benutzer mit dem System vornehmen will (Intention). Die logische Interaktion wird von Alan Dix et al. wiederum in drei Ebenen gegliedert: Die lexikalische Ebene, die syntaktische Ebene und die semantische Ebene [46, S. 545].

Auf der semantischen Ebene wird die Wirkung von Interaktionen auf die internen Datenstrukturen des Systems beschrieben. Hier wird festgelegt welche Informationen ausgegeben werden. Die lexikalische Ebene beschreibt die Repräsentationen von Interaktionselementen. Dies sind z. B. das Aussehen von Elementen am

[1] Eine detaillierte Abhandlung über die Trennung zwischen logischer und physikalischer Interaktion findet sich bei [56].

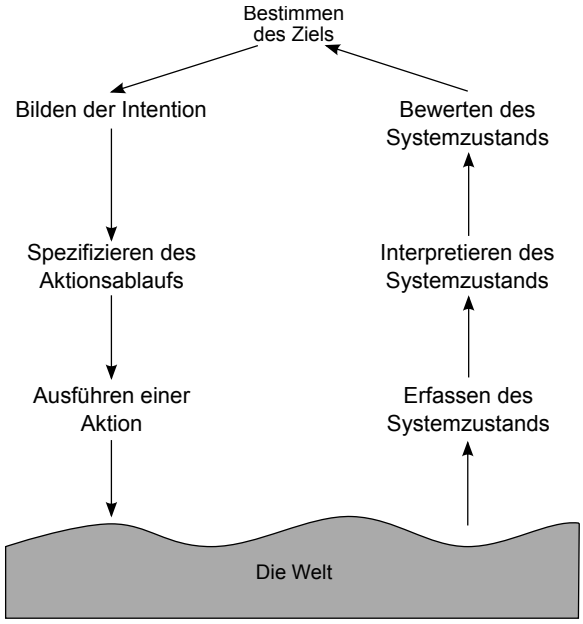


Abbildung 2.3: Interaktionszyklus nach D. Norman [131, S. 47]. Links die Ausführungsphase, rechts die Evaluationsphase.

Bildschirm oder Tasten, welche vom Benutzer gedrückt werden. Auf der syntaktischen Ebene werden Ordnung (Reihenfolge) und Struktur beschrieben. Bei einer grafischen Oberfläche finden sich an dieser Stelle Informationen zum Layout der Elemente, bei Sprachsystemen die Grammatik.

Auf der untersten Ebene findet die physikalische Interaktion statt. Damit bezeichnet man jegliche Interaktion, die ein Benutzer mit Ein- und Ausgabegeräten hat. Beispiele hierfür sind das Drücken einer Maustaste, die Tastatureingabe, oder das Sprechen von Kommandos bei sprachgesteuerten Systemen. Auch die Ausgabe durch verschiedene Geräte fällt hierunter.

Ein Benutzer interagiert mit dem System durch die Interaktionselemente. Dies kann z. B. durch das Betätigen einer Reihe von Knöpfen geschehen, über das Anklicken von Symbolen mit der Maus, über das Berühren eines Touch-Screens oder über eine Spracheingabe.

Das System prüft bei der Eingabe, ob sie syntaktisch korrekt ist. Bestimmte Befehle dürfen z. B. nicht aufeinander folgen, so kann erst nach einer initialen Eingabe eine Undo-Funktion verwendet werden. Wie oben schon erwähnt ist dies

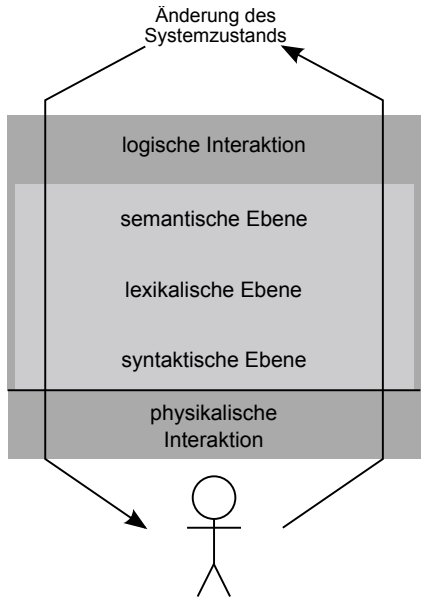


Abbildung 2.4: Interaktionszyklus aus Sicht eines technischen Systems.

auch die Stelle, an der bei Spracheingabe die Worte durch eine Grammatik in Kommandos und andere Worte unterschieden werden. Ebenso können auf den beiden übergeordneten Ebenen Prüfungen der Eingabe erfolgen. Auf der semantischen Ebene lässt sich z. B. überprüfen, ob der Benutzer wirklich ein Datum angegeben hat, wenn er nach einem solchen gefragt wurde. Dazu lässt sich das Wissen über die Art der Daten in dieser Ebene verwenden.

2.1.4 Inhalt, Logik und Form

Insgesamt lässt sich die Interaktion mit einem System in drei Bestandteile trennen: Inhalt, Logik und Form [89]. Unter Inhalt werden üblicherweise Informationen verstanden, welche einem Benutzer vermittelt werden sollen. Inhalte können in verschiedenen Medien wie Sprache, Text oder Bildern vorliegen.

Im Bereich Logik muss zwischen der Interaktionslogik und der Anwendungslogik unterschieden werden. Interaktionslogik beinhaltet die Elemente, welche innerhalb eines Dialogs Teile der Ausgabe beschreiben. Sie enthält einen generischen Anteil, der für eine Modalität fest vorgegeben ist, z. B. der Teil eines Programms,

welcher ein Fenster erzeugt und verwaltet. Weiter enthält sie schematische Anteile, z. B. Widgets[2] einer grafischen Oberfläche. Anwendungslogik bezeichnet Elemente, die auf Funktionalität eines Systems zurückgreifen um z. B. etwas zu berechnen oder in irgendeiner Weise Daten zu verarbeiten.

Die Ausgabe kann analog zur Eingabe in verschiedener Form erfolgen. Schon innerhalb einer Interaktionsart kann es verschiedene Formen geben. So können z. B. Texte in verschiedenen Schriften, Größen und Farben dargestellt werden. Im Bereich des Webdesigns hat sich schon früh herausgestellt, dass eine Trennung von Inhalt und Form sinnvoll ist. So wurde 1994 die erste Version von Cascading Style Sheets (CSS) eingeführt, welche verschiedene Formatierungen und Positionierungsmöglichkeiten erlaubt [208]. In der Regel ist jedoch mindestens die Interaktionslogik des Dialogs mit dem restlichen Programm vermischt, hinzu kommen Teile welche die Eingabeüberprüfungen vornehmen und nicht direkt mit der Interaktionslogik zu tun haben. Bei einem sauberen Entwurf sind zumindest die Interaktions- und die Anwendungslogik von einander getrennt. Die Trennung in die Bestandteile Inhalt, Logik und Form ist schon länger bekannt, auch wenn sich erst um das Jahr 2000 Publikationen, wie von Kerer und Kierda, des Themas annahmen [89].

2.2 Modalitäten

Menschen senden und empfangen laufend Botschaften. So wurde von Watzlawick im Metakommunikativen Axiom postuliert, dass es unmöglich ist *Nichts* zu kommunizieren [202].

Arbeitet man mit einem Computer, so läuft die Interaktion mit dem Gerät zwar anders ab als zwischen Menschen, folgt aber dem gleichen Grundmuster. Der Benutzer nimmt Eingaben am Computer vor, und der Computer reagiert darauf mit Ausgaben. Beides wird über eine Reihe von Kanälen übertragen.

2.2.1 Sinnesmodalitäten

Systemtheoretisch betrachtet handelt es sich bei Sinnesmodalitäten um die Kanäle, über die Nachrichten ausgetauscht werden. In der Medizin wurden diese Kanäle bereits 1837 von Johannes Müller untersucht und als Sinnesmodalitäten bezeichnet [122, 171]. Die Zuordnung geschieht dabei nach dem von Müller formulierten Gesetz der spezifischen Sinnesenergien [19].

Die Sinnesmodalitäten beschreiben aber lediglich die Aufnahme von Informationen beim Menschen.

[2] Widgets sind z. B. Buttons, Labels oder Eingabefelder.

Definition 5 (Sinnesmodalität) *Eine Sinnesmodalität ist ein Empfindungskomplex wie Sehen, Hören, Riechen, Schmecken oder Fühlen. Die Empfindung wird nach dem Sinnesorgan kategorisiert. Der Begriff wird nur für Menschen verwendet.*

Die Zäpfchen der Netzhaut lassen sich zum Beispiel durch Licht und durch Druck stimulieren. Beide Eindrücke geschehen über verschiedene physikalische Mechanismen, doch die Sinnesmodalität *Sehen* ist die selbe.

Am meisten Informationen nimmt der Mensch durch das Sehen auf (ca. 80%). Danach folgt Hören mit 15%, den Rest teilen sich die anderen Sinne [19]. Bei dieser Aufteilung ist zu berücksichtigen, dass die Sinne sich ergänzen, Informationen werden nur bedingt über eine einzelne Sinnesmodalität aufgenommen. Tabelle 2.1 zeigt einen Überblick über alle Sinnesmodalitäten.

	Sinnesmodalität
Hauptkanäle	Sehen (vision)
	Hören (audition)
Haptische Kanäle	Druck, Berührung (tactition)
(auch *tangible*)	Kinästhetische Wahrnehmung (proprioception)
Sonstige	Schmecken (gustation)
	Riechen (olfaction)
	Wärme- und Kälteempfinden (thermoception)
	Schmerz (nociception)
	Gleichgewicht (equilibrioception)

Tabelle 2.1: Überblick über die Sinnesmodalitäten.

Auf die wichtigsten Sinnesmodalitäten wird im Folgenden detaillierter eingegangen.

Sehen Das Sehen ist für fast alle Menschen die wichtigste Art, Informationen aus ihrer Umwelt zu erhalten. Dabei wird Licht durch das Auge aufgenommen und im Gehirn verarbeitet. Das menschliche Auge benötigt Licht, um stimuliert zu werden. Es wandelt das Licht in elektrische Signale um, die es dann über die Nervenstränge an das Gehirn weiterleitet.

Hören Über den Hörsinn nimmt der Mensch Druckunterschiede und Schallwellen wahr und kann darüber hinaus räumlich orten, woher sie kommen.

Das menschliche Gehör kann Frequenzen zwischen 20 Hz und 20 kHz wahrnehmen. Mit zunehmendem Alter verschlechtert sich die Hörfähigkeit insbesondere im oberen Frequenzbereich. Mit Klängen lassen sich zahlreiche Informationen übertragen, wie man am Beispiel der Sprache sehen kann. Bei heutigen Desktop-Systemen werden in der Regel einfache Klänge verwendet, um simple Benachrichtigungen wie Fehler zu übermitteln.

Haptische Kanäle Durch den Tastsinn werden weitere wichtige Informationen übertragen. Man kann z. B. fühlen, ob ein Objekt glatt oder rau, warm oder kalt ist. Darüber hinaus lässt sich auch Druck auf einer Hautfläche wahrnehmen.

Die kinästhetische Wahrnehmung findet durch Rezeptoren in den Muskeln, Gelenken und Sehnen statt und liefert dem Gehirn Informationen über die Position der Körperteile. Diese Wahrnehmung ist für die Benutzung von Werkzeugen sehr wichtig, da sie es ermöglicht, ohne ständige Kontrolle (z. B. durch die Augen) die Gliedmaßen zu positionieren und damit Bewegungen wie Gehen oder Greifen auszuführen.

Bei Blinden verstärkt sich die Wahrnehmung des Tastsinns und des Hörsinns. Beide ersetzen dann zum großen Teil die Informationsübertragung des visuellen Kanals.

Sonstige Sinnesmodalitäten Die Sinnesmodalitäten wie Schmecken und Riechen werden in der Mensch-Maschine Interaktion kaum verwendet, da es aufwendig ist diese Sinne in diesem Kontext zu stimulieren. Es sind erst wenige Geräte entwickelt, die z. B. Düfte zuverlässig erzeugen oder erkennen können. Ein weiteres Problem ist, dass ein einmal erzeugter Duft nicht einfach aus einem Raum entfernt werden kann. Hierfür gibt es noch keine praktikablen Lösungen.

Diese Sinnesmodalitäten könnten jedoch in gewissem Umfang sinnvoll eingesetzt werden, um Informationen über die periphere Wahrnehmung zu vermitteln. In diesem Fall werden Informationen über Temperatur, Börsenkurse oder die Auslastung einer technischen Anlage z. B. durch den Geruch oder Wärme übertragen. Dabei muss zusätzlich noch auf Latenzzeiten bei der Wahrnehmung geachtet werden.

2.2.2 Modalität als Repräsentation von Information

Auch Maschinen verfügen über ein Äquivalent zu Sinnesmodalitäten. Durch Sensoren oder Eingabegeräte können sie Informationen aus ihrer Umwelt aufnehmen. Sie können auch analog zum Menschen auf verschiedene Arten Informationen ausgeben. Sie unterscheiden sich allerdings stark vom Menschen, denn dieser hat nicht alle Ausgabemöglichkeiten von technischen Geräten. Zum Beispiel wird ein Mensch über einen Film sprechen, eine Maschine kann den Film selbst zeigen.

Die Definition allein über die menschlichen Sinne genügt jedoch im Kontext der Interaktion nicht zur Klassifikation. Diese Sicht ist antropozentrisch und lässt sich nicht einfach auf Maschinen übertragen. Was beim Menschen ein Organ und die Nervenzellen zur Verarbeitung der Reize des Organs sind, müsste bei einer Maschine ein ähnlicher Komplex sein. Dies ist jedoch nicht immer der Fall. Zum Beispiel erfolgt die Eingabe eines Textes mit Tastatur oder Stift über die Haptik des Menschen. Für eine Maschine ist die Eingabe über die Tastatur eine Reihe von diskreten Ereignissen. Die Eingabe mit einem Stift ist dagegen eine

kontinuierliche Folge von Eingaben, deren dahinter liegende Geste noch erkannt werden muss. Aus Maschinensicht handelt es sich darum um zwei verschiedene Modalitäten.

Ein anderer Ansatz versucht, Modalitäten über die physikalischen Eigenschaften von Interaktion zu definieren. Card et al. stellten 1990 allein für den haptischen Kanal eine Klassifikation in mehrere Modalitäten auf [29, 30]. In diesem Verfahren werden Eingabegeräte nach mehreren Kriterien beurteilt. Ein Aspekt sind die Koordinaten im Raum, ein weiterer Aspekt sind bspw. Kraftmessungen nach absoluten Werten oder Veränderungen. Zur Verdeutlichung soll eine normale Maus mit mehreren Tasten dienen. Diese besitzt dann zwei Attribute für die Koordinaten im Raum, wobei nur die Veränderungen bei einer Bewegung gemessen werden. Damit ist es möglich die Maus aufzuheben und an einer anderen Stelle wieder abzusetzen, ohne dass sich der Zeiger am Bildschirm bewegt. Für die Tasten werden absolute Werte gemessen, nämlich ob eine Taste gerade gedrückt ist oder nicht. Diese Art der Beschreibung ermöglicht es zwar grundsätzlich, alle möglichen haptischen Eingaben zu erfassen, allerdings lässt sich nur mit viel Aufwand eine Brücke in Richtung des intuitiveren Begriffs der Sinnesmodalität schlagen. Aus Ansätzen wie diesem wurden jedoch allgemeine Beschreibungen für die Eigenschaften von Geräten entwickelt.

Zwischen einem Analogon zu Sinnesmodalitäten und physikalischen Eigenschaften sind verschiedene Abstraktionen denkbar. Es lässt sich eine Unterteilung in drei Ebenen vornehmen, wie bei der logischen Interaktion im Interaktionszyklus (siehe Abschnitt 2.1.3); in die lexikalische, die syntaktische und die semantische Ebene [126]. Die lexikalische Ebene beschreibt physikalische Eingabeereignisse wie einen Tastendruck oder Mausklick, die syntaktische Ebene dient zur Kontrolle, ob Eingaben in einer vorgegebenen Reihenfolge geschehen. Die semantische Ebene dient der Prüfung, ob die gewählte Aktion im aktuellen Zusammenhang sinnvoll ist.

Nigay und Coutaz beschrieben 1995, dass die Kombination eines Geräts mit einer Interaktionssprache eine *interaction technique* ergibt. Diese entspricht schon sehr dem bisher erweiterten Modalitätsbegriff [127]. Eine Interaktionssprache beschreibt dabei die verschiedenen Ebenen der Interaktion mit einem Gerät. So enthält eine Interaktionssprache für Spracheingabe Schlüsselworte, eine Grammatik und Regeln, um die Eingabe zu prüfen.

Daraufhin beschrieb Nigay zusammen mit Vernier 2001 den Begriff Modalität als eine Kombinationen einer Interaktionssprache und eines Geräts mit der Unterscheidung in Ein- und Ausgabemodalität [201]. Der Begriff der *interaction technique* wurde durch den gebräuchlicheren Begriff der Modalität ersetzt. Die Trennung in Ein- und Ausgabemodalitäten ist von Bedeutung, denn durch sie lassen sich unabhängige Sprachen für unterschiedliche Teile eines Systems einsetzen. Eine Beschreibungssprache für die Eingabe über Maus und Tastatur ist z. B. unabhängig von einer Beschreibung für eine grafische Ausgabe auf einem Monitor. Eine Modalität ist nicht allein das Gerät oder das Medium, denn selbst für eine einzelne Modalität kommt es auf die Interaktionssprache an. So lassen sich durch

Signaltöne und natürliche Sprache Informationen über das selbe Gerät und das selbe Medium übertragen, doch die Informationen sind deutlich verschieden repräsentiert. In derselben Arbeit finden sich auch Definitionen von Modalität und Modus. Vernier und Nigay sehen eine Modalität als einen Typ eines Kommunikationskanals in Kombination mit der Art, in der ein Konzept ausgedrückt oder verstanden wird [3]. Diese Definition baut auf den obigen Vorarbeiten von Modalität als Kombination von Interaktionssprache und Gerät auf. Ein Modus ist ein Zustand, welcher die Art bestimmt, in der Informationen verstanden werden.

Bernsen stellte fest, dass eine Modalität auch als Art der Informationsrepräsentation gesehen werden kann. Dazu muss die Information physikalisch dargestellt werden [15]. In Bernsens Arbeit werden verschiedene Arten der Informationsrepräsentation gesammelt, um aus ihrer Kombination die Menge aller möglichen Basismodalitäten zu schaffen. Für diese Arbeit ist ein solcher Ansatz allerdings ebenso wenig zielführend wie eine Auflistung aller möglichen physikalischen Interaktionsmöglichkeiten. Es genügt, verschiedene Arten von Modalitäten unterscheiden zu können.

Wie in Abschnitt 2.1.4 zur Trennung von Inhalt, Logik und Form bereits bemerkt, besitzt eine Modalität dabei eine Richtung.

Im Folgenden wird der Begriff *Modalität* definiert. Analog zu den Sinnesmodalitäten kann man auch von Interaktionsmodalitäten sprechen, doch wird in dieser Arbeit der kürzere Begriff verwendet.

Definition 6 (Modalität) *Eine Modalität ist eine Art der Informationsrepräsentation für Mensch oder Maschine, in der Information in einer bestimmten Form physikalisch dargestellt wird. Modalitäten können durch die Art der Informationsrepräsentation unterschieden werden.*

Die Art der Informationsrepräsentation entspricht der oben beschriebenen Interaktionssprache in Kombination mit den benötigten Geräten. Zwei Beispiele sollen die Unterschiede der Informationsrepräsentation verdeutlichen. Ein System erlaubt es, Text einzugeben. Eine Tastatur benötigt eine Interaktionssprache, über die z. B. Kürzel oder einzelne Tasten zur Bedienung einer Anwendung definiert sind. Die Eingabe über einen Stift kann durch eine Sprache zur Erkennung von Buchstaben, ähnlich wie eine Tastatur, genutzt werden. Soll dagegen eine Unterschrift kalligraphisch eingegeben werden, so muss die Sprache den Pfad anstelle von einzelnen Buchstaben beschreiben können. Als weiteres Beispiel soll die Ausgabe von akustischen Signalen und natürlicher Sprache dienen. Hier unterscheiden sich die Sprachen zur Repräsentation der auszugebenden Informationen noch stärker voneinander.

Nicht immer sind die Unterschiede so klar wie in den hier gewählten Beispielen. Bei einer Stifteingabe kann z. B. zusätzlich der ausgeübte Druck gemessen

[3] siehe auch die Vorarbeit von Coutaz, Nigay und Salber [38]

werden. Auch wenn sich hier beide Modalitäten durch die Eigenschaften der Geräte unterscheiden, könnte man diese zusammenzufassen. Für eine ausführliche Abgrenzung dieser Modalitäten sei auf Bernsens Arbeit verwiesen [15, 16].

2.2.3 Multimodalität

Modalitäten treten in verschiedenen Ausprägungen auf. Falls zur Ein- oder Ausgabe nur eine einzelne Modalität zum Einsatz kommt, nennt man dies *unimodal*. Interessant wird es, wenn mehrere Modalitäten zum Einsatz kommen.

Bernsen definiert Multimodalität lediglich über die Ausgabemodalität. Ein System ist multimodal, falls Informationen auf verschiedene Arten repräsentiert und ausgegeben werden. Dies kann gleichzeitig oder sequentiell geschehen [15].

Nigay und Vernier sehen ein System als multimodal an, falls es verschiedene Interaktionssprachen zum Einsatz bringt. Dabei kann ein einzelnes Gerät verwendet werden, so dass z. B. die Anzeige von Text, Bild oder Animation auf einem Bildschirm ausreichend ist. Ein einzelnes Gerät kombiniert in diesem Fall mehrere Interaktionssprachen, man kann deshalb von verschiedenen Modalitäten sprechen [201].

In dieser Arbeit sollen jedoch nicht nur die Ausgabemodalitäten sondern auch die Eingabemodalitäten betrachtet werden. Die folgende Definition berücksichtigt deshalb beide Arten von Modalität. Multimodalität kann bei einem System dann bei der Eingabe, der Ausgabe oder beidem auftreten.

Definition 7 (Multimodalität) *Treten in einem Dialog mehrere Ein- oder Ausgabemodalitäten parallel oder sequentiell auf oder überlappen sie sich im zeitlichen Ablauf, so wird dies multimodal genannt.*

Falls während der Durchführung einer Interaktion die Modalität gewechselt wird oder falls zwei Modalitäten vorhanden sind, die sonst unabhängig genutzt werden, die in diesem Fall aber überlappend für verschiedene Teile einer Interaktion genutzt werden, so heißt dies *transmodal*. Eine Eingabe, bei der ein Benutzer gleichzeitig auf ein Objekt deutet und dazu ein Kommando durch Sprache sagt, ist ein Beispiel für Transmodalität. Sowohl Sprache als auch Gestik können als eigenständige Modalitäten genutzt werden, hier sind sie jedoch zu einer Interaktion verquickt.

Nur weil ein System multimodale Interaktion anbietet, bedeutet dies aber nicht, dass Benutzer sofort multimodal damit arbeiten. Vielmehr ermöglicht Multimodalität es Benutzern die für sie und die aktuelle Aufgabe geeignete Modalität einzusetzen. Damit erhöht sich nach einer Einarbeitung- und Gewöhnungsphase die Effizienz der Benutzer [143].

2.2.4 CARE-Eigenschaften

Coutaz et al. entwickelten 1995 die sogenannten *CARE properties* [40]. CARE steht dabei für Complementarity, Assignment, Redundancy und Equivalence. Diese Eigenschaften dienen der formalen Beurteilung von Interaktionsmodellierung für Modalitäten. Ein System wird als Zustandsautomat betrachtet, dessen Kanten Interaktionen durch Modalitäten sind.

- Complementarity (Komplementarität): Falls in einem Zeitraum mehrere Modalitäten genutzt werden müssen, um den Zustand eines Systems zu ändern, so heißen sie komplementär zueinander.
- Assignment (Zuweisung): Falls nur eine Modalität einsetzbar ist, um den Zustand eines Systems verändern zu können, so ist diese Modalität der Zustandsänderung zugewiesen.
- Redundancy (Redundanz): Wenn zwei Modalitäten über die gleiche Ausdrucksmächtigkeit verfügen, über beide der Zustand des Systems identisch verändert werden kann und sie im selben Zeitraum genutzt werden können, dann sind sie redundant zu einander.
- Equivalence (Äquivalenz): Wenn durch zwei Modalitäten das gleiche Ziel erreicht wird, sind sie äquivalent.

Die CARE-Eigenschaften stellen einen formalen Rahmen dar, innerhalb dessen Aussagen über multimodale Systeme getroffen werden können. Man kann mit ihnen zum Beispiel überprüfen, ob manche Aktionen nur über eine bestimmte Modalität erreichbar sind, ob äquivalente Pfade vorhanden sind oder mehrere Modalitäten gemeinsam genutzt werden müssen.

2.2.5 Wichtige Modalitätskategorien

Bisher war in dieser Arbeit von grafischen Oberflächen (engl. *graphical user interfaces*, GUIs) die Rede, doch nach der obigen Definition können zwei verschiedene GUIs bereits verschiedene Modalitäten sein. Damit es einfacher ist, über Gruppen von ähnlichen Modalitäten zu sprechen, bilden diese eine Modalitätskategorie. Dabei dürfen sich Repräsentanten einer Kategorie nur in wenigen Attributen voneinander unterscheiden.

Definition 8 (Modalitätskategorie) *Eine Modalitätskategorie fasst Modalitäten zusammen, die sich sehr ähnlich sind.*

Bei der Einteilung in Kategorien haben sich im Lauf der Zeit verschiedene Gruppen herausgebildet. Dix et al. beschreiben Interaktionsstile, die sich weitgehend

als Kategorien anbieten [46, S. 136]. Grafische Oberflächen sind in zwei Varianten, Point and Click und WIMP (Windows, Icons, Menus, Pointers) aufgeführt. Sprache zur Bedienung von Systemen findet sich als Natural Language, wobei heutige Sprachsysteme zwischen natürlicher Sprache und einfacheren Systemen mit Sprachkommandos unterscheiden. 3D ist eine Kategorie, die in den letzten Jahren insbesondere durch Spiele eine größere Verbreitung gefunden hat.

Bei Dix et al. finden sich in den Interaktionsstilen allerdings auch Elemente, die weniger als Kategorie geeignet sind. Diese Elemente beschreiben vielmehr Abläufe oder Verhalten von Interaktionen und sind nicht immer an Modalitäten gebunden. Auf sie passt die Bezeichnung Interaktionsmuster (*interaction patterns*), die von Jennifer Tidwell, die eine solche Mustersammlung zusammengestellt hat, geprägt wurde [46, 188, S. 284ff]. Beispiele für Interaktionsmuster sind Menüs, Wizards und Formulare. Menüs können unter anderem grafisch oder durch Sprache realisiert werden, und sind modalitätsunabhängig. Wizards sind Fragedialoge, die einen Benutzer durch eine Reihe von Fragen führen. Formulare sind in grafischen Oberflächen, wie Buchungssystemen, im Einsatz.

Bei Benutzerschnittstellen existieren auch Stile zur unterschiedlichen Repräsentation. Bei grafischen Oberflächen kann ein Benutzer z. B. auffälligere Farben bevorzugen, ein anderer legt Wert auf dezente Töne und passende Schriften. Bei Sprache werden dagegen verschiedene Stimmlagen und Tempi favorisiert. Allerdings spricht man meist nicht von Stilen, sondern bei grafischen Oberflächen von Präsentationsvorlagen (engl. *Templates*) oder dem *Look & Feel*. Dabei werden einheitliche Darstellungen wie die genannten Farben, Schriften oder Stimmlagen für Benutzerschnittstellen beschrieben, so dass sie für verschiedene Anwendungen eingesetzt werden können. Ein Beispiel einer solchen Beschreibungssprache sind die Cascading Style Sheets (CSS) [208].

Im Folgenden wird ein kurzer Überblick über drei etablierte Modalitätskategorien und einige weitere Modalitäten gegeben.

Grafische Benutzerschnittstellen (GUIs)

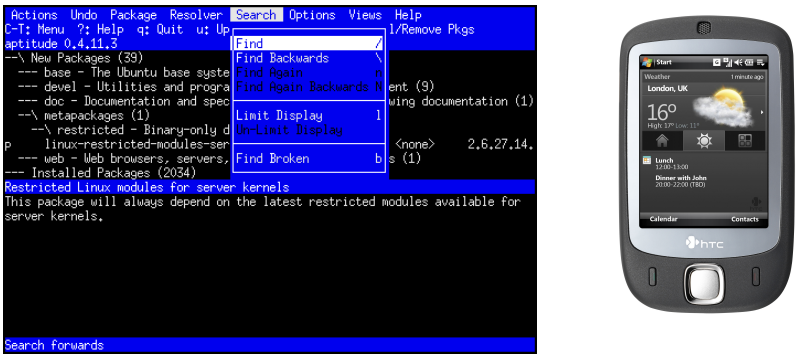
Eine grafische Benutzerschnittstelle (GUI[4]) ist eine Modalitätskategorie, die ein haptisches Eingabegerät und einen Bildschirm verwendet. Streng genommen müsste die Kategorie als grafisch-haptisch bezeichnet werden, doch der kürzere Ausdruck hat sich durchgesetzt. Frühe GUIs waren Text User Interfaces, als Eingabegerät kamen klassisch Tastatur und Maus zum Einsatz. Mittlerweile finden sich Varianten die mit Stiften bedient werden. In Abbildung 2.5a sieht man eine aktuelle Anwendung zur Paketverwaltung unter Linux [20].

Die klassische GUI ist jedoch ein fensterbasiertes System, das bei allen gängigen Betriebssystemen wie Windows, MacOS oder Unix als Interaktionsschnittstelle

[4] engl. *graphical user interface*

Anwendung findet. Dieser Interaktionsstil wird auch WIMP genannt, da er Fenster, Icons, Menüs und ein Zeigegerät (Maus) verwendet.

GUIs sind die derzeit am meisten verbreiteten Benutzerschnittstellen im PC-Bereich. Sie eignen sich besonders gut für Aufgaben, bei denen bildlich/textuell gearbeitet werden muss, oder zum Erstellen von grafischen Inhalten wie z. B. dem Nachbearbeiten von Fotos.



(a) Debian Aptitude (b) HTC Touch

Abbildung 2.5: Zwei GUIs, links eine Text-basierte Oberfläche zur Paketverwaltung, rechts ein Mobiltelefon mit Touchscreen von HTC (`http://www.htc.com`).

Wichtige Neuerungen sind zoombare GUIs, wie von Perlin und Meyer 1999 vorgestellt [152], sowie die Manipulation von GUIs durch zwei Mäuse [64]. In den letzten Jahren finden auch immer mehr Geräte Verbreitung, die berührungsempfindliche Oberflächen zur Eingabe verwenden. Diese können mit einem oder mehreren Fingern bedient werden, wie z. B. die Multitouch-Oberfläche Microsoft Surface [111]. In Abbildung 2.5b ist der HTC Touch abgebildet, ein Mobiltelefon, das über einen Touchscreen gesteuert werden kann.

Sprachbedienschnittstellen

Sprache wird je nach Art der Anwendung unterschiedlich beschrieben, wodurch es mehrere Sprachmodalitäten geben kann. Natürliche Sprache kann genauso zum Einsatz kommen wie eine reine Kommandosprache. Je nach Einsatzzweck kann eine Sprachbedienschnittstelle eine sehr natürliche Form der Interaktion sein. Doch Sprache ist durch ihre Vieldeutigkeit nicht die einfachste Art der Interaktion [45, S. 118]. Für Untermengen von Sprachen, die sich auf Anwendungs-

gebiete beschränken, existieren allerdings Systeme mit hohen Erkennungsraten. Sprachbedienschnittstellen werden in der Regel als Sprachdialogsysteme (SDS) realisiert.

Die Möglichkeit, über natürliche Sprache mit Maschinen zu kommunizieren, existiert seit den 1990er Jahren. Davor gab es lediglich eine sehr eingeschränkte Form der Kommunikation, indem ein Menü vorgelesen wurde und der Benutzer dann über eine Taste eine Auswahl treffen konnte. Dies ist nach wie vor noch die häufigste Art, einen Sprachdialog mit einer Anwendung zu führen, wie das folgende Beispiel zeigt [43]:

> *System:* Bitte geben sie ihre PIN ein und drücken sie anschließend die Sterntaste.
>
> *Anrufer:* 1 2 3 4 *
>
> *System:* Hier ist ihr Anrufbeantworter. Sie haben keine Nachrichten. Sie befinden sich im Hauptmenü. Zum Abfragen ihrer Nachrichten drücken sie die 1. Die Einstellungen ihres Anrufbeantworters ändern sie über die 3. Für weitere Leistungsmerkmale drücken sie die 4. [...]

Für einfache Anwendungen wie Anrufbeantworter genügt diese Form des Dialogs. Komplexere Einsätze bleiben Anwendungen bei dieser Form des Dialogs aber verwehrt. Der Vorteil ist die einfache Realisierungsmöglichkeit dieser Dialogsysteme. Das Verfahren, die Eingaben des Benutzers über Tonwahl auszuwerten, hat sich in der Praxis bewährt, da es robust ist und Unsicherheiten bei der Erkennung der Eingaben vermeidet.

McTear unterscheidet in seinem Überblickartikel zwischen drei Arten von Systemen: zustandsbasierte, frame-basierte und agenten-basierte Systeme [104]. Das obige Beispiel ist klar zustandsbasiert. Frame-basierte Systeme arbeiten mit Templates und erlauben etwas mehr Freiheiten. Agenten-basierte Systeme ermöglichen natürlichsprachliche Dialoge und das Führen komplexer Dialoge. Das folgende Beispiel von 1997 zeigt einen natürlichen Dialog mit einem solchen agenten-basierten System [168].

> *User:* I'm looking for a job in the Calais area. Are there any servers?
>
> *System:* No, there aren't any employment servers for Calais. However, there is an employment server for Pas-de-Calais and an employment server for Lille. Are you interested in one of these?

Das agenten-basierte System ist in der Lage zusätzlich zur abschlägigen Antwort Alternativen anzubieten und den Dialog mit dem Benutzer weiterzuführen.

Tangible User Interfaces (TUIs)

Eine Modalitätskategorie, die in den letzten Jahren mehr Verbreitung findet, sind *Tangible User Interfaces* (TUIs), die manchmal auch als *Reality based User Interfaces* (RBIs) bezeichnet werden. TUIs sind Benutzerschnittstellen, bei denen einzelne physikalische Objekte vom Benutzer manipuliert werden können. Die Objekte sind dabei Repräsentanten von digitaler Information, mit der ein Benutzer durch sie interagiert.

Wie Rekimoto 2008 beschrieben hat, ist der größte Unterschied von TUIs zu GUIs, dass die manipulierbaren Benutzerschnittstellen organischer und für den Anwender gewohnter sind [165]. Auch Jacob sieht einen Trend zu TUIs und zeigt auf, wie sie sich allmählich in den alltäglichen Geräten verbreiten [81].

Ein Beispiel für ein TUI ist *Urban Planning* (URP), das von Underkoffler und Ishii 1999 vorgestellt wurde [192]. URP ist ein experimentelles System zur Städteplanung, das in Abbildung 2.6 gezeigt wird. Ein Benutzer kann Gebäudemodelle aus Drahtgestellen bauen und auf eine Oberfläche stellen. Je nach eingestellter Tageszeit werden die jeweiligen Beleuchtungsverhältnisse auf der Oberfläche dargestellt, indem die Gebäude Schatten werfen. Durch die Veränderung der Tageszeit kann ein Benutzer sehen, welche Gebäudeteile abgeschattet werden. Zusätzlich lässt sich die Transparenz der Schatten mit Hilfe eines Stabs verändern. Es lässt sich einstellen, ob die Wände der Gebäude aus Glas oder lichtundurchlässigem Material sein sollen. Dementsprechend passt das System die Schatten an. Durch ein Wind-Werkzeug, welches auch ein Objekt ist, das auf die Oberfläche gelegt wird, erscheinen Pfeile unterschiedlicher Länge auf der Oberfläche. Dadurch kann erkannt werden, wie der Wind um die Gebäude streichen wird, wobei die Windrichtung durch einfaches Drehen des Wind-Werkzeugs verändert werden kann. Die Pfeillänge gibt dabei die Stärke des Windes an. Der Benutzer kann jederzeit die Gebäude anders positionieren um so ihre optimale Anordnung zu bestimmen.

Während frühere TUIs statischer Natur waren und Eingaben durch Verschieben oder Drehen von Objekten geschah, lassen sich TUIs der zweiten Generation dadurch nutzen, dass man ihre Materialien direkt verformten kann [80]. Ein Beispiel für diese Art von TUI ist SandScape, ein System, in dem der Benutzer eine Landschaft aus Sand verformt und gleichzeitig das Ergebnis der Landschaftsberechnungen auf dem Sand durch eine Projektion angezeigt bekommt [79]. Die Benutzeroberfläche des Systems ist in Abbildung 2.7 zu sehen. Zusätzlich wird in diesem Fall noch eine Ansicht von der Position der Finger des Benutzers auf die Seitenwand projiziert.

Weitere Modalitäten

Es gibt noch eine Reihe von weiteren Modalitäten, die derzeit noch nicht weit verbreitet sind.

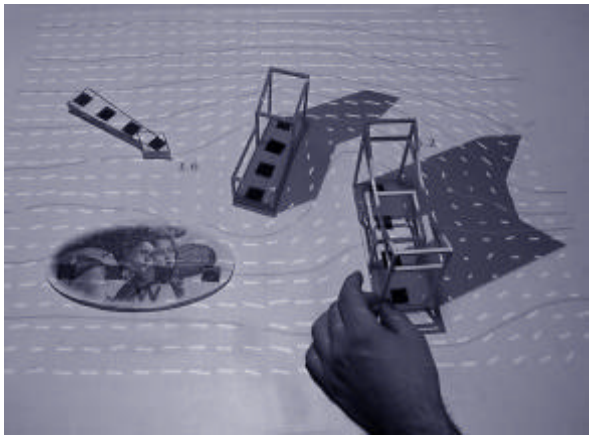


Abbildung 2.6: Planung mit URP. Links unten das Wind-Werkzeug (Bild aus [191]).

Interaktion in virtuellen Räumen wurde bereits 1997 mit VRML standardisiert [78], doch existieren nur wenige Anwendungen in Spezialgebieten, die Interaktion in dreidimensionalen virtuellen Räumen nutzen.

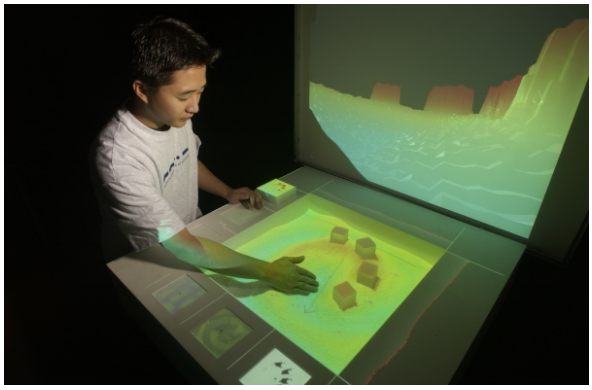


Abbildung 2.7: Landschaftsplanung mit SandScape. Der Benutzer verformt den Sand und sieht zugleich die projezierten Ergebnisse (Bild aus [79]).

Der Bereich der *Augmented Reality* [109], überlagert Informationen durch Projektion und ähnliche Verfahren. Augmented Reality findet Anwendung in Bereichen, in denen beispielsweise Techniker unterstützt werden. Diese können beispielsweise durch eine besondere Brille Anweisungen zur Wartung von Motoren abrufen, die sie sonst zeitintensiver suchen müssten [182]. In Fahrzeugen kommt diese Modalität immer mehr zum Einsatz, z. B. serienmäßig in der Serie 5 von BMW [18].

Im Bereich der taktilen Ausgabe sind seit einigen Jahren Braillezeilen für blinde Benutzer im Einsatz, die sich durch Fortschritte bei der Mechatronik mittlerweile günstiger und leichter herstellen lassen. Ähnliche Verfahren sind im Bereich Forcefeedback zum Einsatz gekommen. So existieren z. B. Mäuse die durch verschiedene Vibrationen Ausgaben an den Benutzer erlauben und auch experimentell in Benutzerschnittstellen zum Einsatz kommen [50].

Die Überblicksarbeit von Hornecker und Buur zeigt, dass sich im Bereich der Tangible Interaction weitere Modalitäten ausbilden [69]. Dies sind unter anderem Interaktion mit dem ganzen Körper (*full body interaction*), d. h. die Möglichkeit den gesamten Körper für Interaktionen zu verwenden, und Interaktion durch die räumliche Anordnung von Objekten vorzunehmen [175]. Vermutlich wird sich im Lauf der Zeit eine neue Modalitätskategorie bilden, da diese Form der Interaktion nicht mehr zu TUIs passt.

Für den breiten Einsatz geeignet ist die Eingabe durch die Gestik der Hände oder einzelner Finger [44]. Mausgesten sind schon länger vorhanden und bereits in verschiedenen GUIs im Einsatz. Manche Anwendungen lassen sich auch nachträglich durch Plugins erweitern, so dass sie Gesten erkennen. Durch ein entsprechendes Plugin lässt sich bspw. ein Browser so erweitern, dass ein Tab geschlossen bzw. geöffnet werden kann oder man durch die History navigieren kann [7]. Die Eingabe von Gesten lässt sich schnell von Benutzern erlernen und im Alltag anwenden.

Durch die Blickrichtung kann ein Benutzer auf einfache Art Objekte auswählen. Dazu lassen sich die Bewegungen der Augen verfolgen, wie es schon manche Kameras anbieten [27]. Der Einsatz innnerhalb einer Kamera dient zur Fokussierung des Motivs und erlaubt ein schnelleres Arbeiten. Weitere Ansätze bestimmen die Blickrichtung direkt über eine Kamera, ohne dass der Benutzer seine Augen in einem festen Winkel auf Meßgeräte richten muss [203, 186].

Eine Modalität, die es bereits seit über zehn Jahren gibt, und die in den letzten Jahren eine größere Verbreitung gefunden hat, ist direkte Eingabe durch *Touch* bzw. *Multitouch*. Touch bedeutet, dass durch die Berührung eines Displays direkt mit Objekten gearbeitet werden kann. Bei Multitouch lassen sich mehrere Finger gleichzeitig einsetzen. Touch ist mit der Eingabe durch Tastaturen verwandt und ein Spezialfall einer GUI. Vorteile von Touch sind die einfache Interaktion und dass keine zusätzlichen Eingabegeräte wie eine Maus benötigt werden. Nachteile sind die geringere Präzision der Eingaben, Ermüdung bei langer Benutzung und Verschmutzung des Displays [207].

2.3 Modelle

In den vorherigen beiden Abschnitten dieses Kapitels wurde der Begriff der Interaktion eingeführt und die Art, in der Benutzer mit Anwendungen interagieren, geklärt. Im Folgenden wird der Begriff des Modells betrachtet. Im Folgenden wird der Begriff des Modells untersucht.

Bei der Erstellung von Anwendungen soll systematisch und geplant vorgegangen werden. Dies ist Thema des Bereichs Software Engineering [180]. Ein Modell ist eine abstrakte Beschreibung der Wirklichkeit, anhand derer Aussagen über das Verhalten, Daten und die Zusammensetzung aus Komponenten der Anwendung getroffen werden können.

Die modellbasierte Entwicklung von interaktiven Systemen, um die es in dieser Arbeit geht, zielt darauf ab Modelle mit einem hohen Abstraktionsgrad zu identifizieren. Sie erlauben es, ein System auf einer semantisch orientierten Ebene zu spezifizieren und zu analysieren [146]. Aus einer Spezifikation kann eine Anwendung generiert werden.

Deklarative modellbasierte Ansätze bieten nach Nunes drei Vorteile [132]:

1. Sie bieten eine abstrakte Sicht auf die Benutzerschnittstelle und erlauben es damit, sich erst später mit primitiveren Details zu beschäftigen. Dabei fördern sie die Exploration und Innovation.
2. Sie ermöglichen Methoden, die systematisches Design unterstützen.
3. Sie liefern die Infrastruktur, die zur Automatisierung der Prozesse der Gestaltung und Erzeugung von Benutzerschnittstellen benötigt wird.

Für interaktive Anwendungen spielen hauptsächlich das Verhalten der Anwendung gegenüber dem Benutzer und die Interaktionsmöglichkeiten des Benutzers eine Rolle.

Modelle lassen sich für verschiedene Zwecke einsetzen. Zunächst wird vorgestellt, wie Software modellgetrieben entwickelt wird, danach, wie Modelle direkt ausgeführt werden können.

Model Driven Software Development

Im Software Engineering wurde über mehrere Jahre der Einsatz von Modellen zur Codegenerierung erforscht. Dies geschah, da im Lauf der Zeit die zu entwickelnden Systeme immer komplexer wurden und Änderungen an ihnen sich darum sowohl während der Entwicklung, als auch in der Wartungsphase immer aufwändiger gestalteten. Erschwerend kommt hinzu, dass viele Plattformen wie

Java und .NET, sowie eine Vielzahl von Endgeräten wie Desktops, Netbooks und Mobiltelefone unterstützt werden sollen.

Im Software Engineering wird in den meisten Vorgehensweisen mit der Spezifikation der Anwendung begonnen, bei der häufig Modelle eingesetzt werden. Anstatt diese lediglich für die Dokumentation und zur Kommunikation zwischen Entwicklern einzusetzen, werden beim *Model Driven Software Development* (MDSD) die Modelle zur Erzeugung von Quellcode eingesetzt [99]. Ein anderer Name für MDSD ist *Model Driven Development* (MDD[5]). Dieser Begriff geht auf das Feld der *Model Driven Architecture* zurück, welches sich primär mit dem Modellieren von Anwendungen befasst [135]. Die Object Management Group ist ein internationales Konsortium, welches dazu eine Reihe von Standards erstellt hat.

Zur Spezifikation von Anwendungen existieren eine Reihe von Standards, der bekannteste davon ist die *Unified Modeling Language* (UML) [142]. Dabei kann man auf verschiedenen Abstraktionsebenen modellieren. Die UML kann hard- und software-unabhängige Modelle ebenso wie hardwarenahe Timing-Diagramme, beschreiben.

Beim MDD-Ansatz bilden Modelle, die unabhängig von der Hardware-Plattform sind, das sogenannte *Platform Independent Model* (PIM). Durch Transformationen wird daraus das *Platform Specific Model* (PSM) einer Anwendung für eine Zielplattform erzeugt. Durch den Transformationsschritt werden die Anpassungen an die Gegebenheiten der Zielplattform vorgenommen. Von einer Anwendung zum Abspielen von Musik werden zum Beispiel ein Modell des Verhaltens und ein Klassendiagramm erstellt. Diese bilden zusammen das PIM. Durch eine Transformation wird Code für eine Desktopanwendung in der Sprache C++ erzeugt. Durch eine weitere Transformation wird aus dem PIM Code für ein Mobiltelefon in C# erzeugt. Diese beiden erzeugten Programme sind plattformabhängig und bilden somit jeweils ein PSM. An ihnen müssen nur noch wenige Anpassungen vorgenommen werden, damit der Programmcode auf den Zielplattformen ausführbar ist.

2.4 Fallbeispiele

In den nächsten Kapiteln werden die Eigenschaften von Modellen, die Erstellung von multimodalen Benutzerschnittstellen und andere Aspekte vorgestellt.

Anhand ausgewählter Fallbeispiele werden die theoretischen Aspekte erläutert. Als Beispiele wurden ein Getränkeautomat und ein MP3-Player ausgewählt. Beide konzentrieren sich auf jeweils eine kleine Menge von möglichen Interaktionen. Sie werden hier kurz eingeführt, um später auf sie zurückgreifen zu können.

[5] Name von der Object Management Group markenrechtlich geschützt

2.4.1 Getränkeautomat

Getränkeautomaten wie in Abbildung 2.8a gibt es in vielen größeren Einrichtungen. Manche Firmen bieten ihren Angestellten kostenlos warme Getränke an, in diesem Beispiel soll jedoch für die Leistungen bezahlt werden. Da für diese Arbeit der Bau eines solchen Geräts nicht in Frage kommt, wird ein Automat mit einer softwarebasierten Benutzerschnittstelle verwendet. Die nachgebildete Oberfläche ist in der Abbildung 2.8b zu sehen.

(a) Kaffeeautomat (b) Automat in Software

Abbildung 2.8: Links ein Kaffeeautomat, rechts die Benutzerschnittstelle eines Automaten als Anwendung nur in Software (Bild links von `http://designnation.de`).

In diesem Fallbeispiel hat der Automat den Vorteil, dass er nicht vereinfacht werden muss. Die gesamte Funktionalität lässt sich modellieren und in das Beispiel übernehmen. Das Fallbeispiel wird anhand des realen Automaten beschrieben, statt Tasten können jedoch Buttons einer GUI oder Sprachbefehle verwendet werden.

Zu Beginn begrüßt der Automat jeden Benutzer. Bei einem realen Automaten geschieht dies durch eine Anzeige. Ein Benutzer kann Preise abfragen, indem er die verschiedenen Tasten des Geräts drückt. Die Preise werden kurz angezeigt, danach kehrt der Automat zur Begrüßung zurück.

Sobald ein Benutzer Geld einwirft, kann er den Preis nicht mehr abfragen. Der Automat zeigt das Guthaben an und lässt Milch und Zucker zu den Getränken wählen. Sobald ein Getränk ausgewählt wird, beginnt der Automat mit der Herstellung. Falls das Guthaben nicht ausreicht, wartet der Automat darauf, dass der Benutzer noch mehr Geld einwirft. Der Benutzer kann vor der Herstellung des Getränks jederzeit Abbrechen und bekommt sein Geld zurück.

Nachdem ein Getränk hergestellt wurde, gibt der Automat das Rückgeld aus, wartet kurz auf die Entnahme des Bechers und kehrt dann wieder zur Begrüßung zurück.

2.4.2 MP3-Player

MP3-Player sind heute sehr weit verbreitet. Es gibt sie als separate Geräte oder integriert in größeren Geräten wie Mobiltelefonen, Autoradios oder DVD-Abspielgeräten. Auch als Programm gibt es viele Variationen für eine Vielzahl von Plattformen. In Abbildung 2.9a ist ein dediziertes Abspielgerät dargestellt. In Abbildung 2.9b sieht man den Media Player Classic, ein weit verbreitetes Programm zum Abspielen von MP3s für Microsoft Windows.

File View Play Navigate Favorites Help
2. Cake - Short Skirt, Long Jacket.mp3
3. Delgados - the-light-before-we-land.mp3
4. Valkyris.mp3
Playing

(a) Apple Ipod (b) Media Player Classic

Abbildung 2.9: Zwei MP3-Player: Links Apple Ipod, rechts Media Player Classic (`http://mpc-hc.sourceforge.net/`).

Für das Fallbeispiel findet eine Vereinfachung statt. Auf Bedienelemente wie Aufnahmefunktionen, grafische Visualisierung der Musik oder Equalizer wird verzichtet.

Der MP3-Player besitzt eine Playlist, es ist möglich die Lautstärke einzustellen und Musik abzuspielen. Die Playlist bildet eine Schleife und ist der Einfachheit halber nicht veränderbar. Nach dem Einschalten begrüßt der MP3-Player den Benutzer durch eine kurze Nachricht und ist danach für das Abspielen von Musik bereit. Tabelle 2.2 zeigt die verschiedenen Aktionen, die mit dem MP3-Player möglich sind und was sie bewirken. Zunächst wird keine Musik gespielt, erst nach der Eingabe des Befehls `play` beginnt die Anwendung damit. Befehle sollen hier abstrakt von der eigentlichen Benutzerschnittstelle beschrieben werden. Sie

können als Taste an einem Gerät, Button in einer GUI oder Sprachbefehl realisiert werden.

Aktion	Auswirkung
`play`	Startet das Abspielen eines Lieds.
`pause`	Unterbricht das Abspielen.
`stop`	Unterbricht das Abspielen und setzt die Abspielposition auf den Anfang des Lieds zurück.
`inc_volume`	Erhöht die Lautstärke.
`dec_volume`	Senkt die Lautstärke.
`next`	Ändert das aktuelle Lied auf das nächste der Playlist. Im Abspielmodus wird das Lied sofort gestartet.
`previous`	Wie `next`, nimmt aber das vorherige Lied der Playlist.

Tabelle 2.2: Mögliche Aktionen mit dem MP3-Player.

2.5 Fazit

In diesem Kapitel wurden die grundlegenden Begriffe eingeführt und definiert. Die Arten der Interaktion wurden geklärt und dabei die Begriffe der Sinnesmodalität, Modalität, Multimodalität und Modalitätskategorie eingeführt.

Modelle nach dem Model Driven Software Development wurden als Grundlage für die Erzeugung von plattformspezifischen Modellen aus generischeren, plattformunabhängigen Modellen vorgestellt.

Zum Schluss wurden Fallbeispiele vorgestellt, die in der Arbeit immer wieder aufgegriffen werden, um Ansätze zu veranschaulichen.

Die folgenden Kapitel verwenden die eingeführten Begriffe, insbesondere den Modalitätsbegriff. In der Einleitung wurde in der Problemstellung aufgezeigt, dass die Vielfalt an vorhandenen Modalitäten die Erstellung von multimodalen Anwendungen erschwert. Durch den Einsatz geeigneter Modelle lassen sich Verfahren des Model Driven Software Development anwenden, welche die Erstellung der multimodalen Anwendungen erleichtern.

3 Klassifikation bestehender Ansätze

> During the process of stepwise refinement, a notation which is natural to the problem in hand should be used as long as possible.
>
> *Niklaus Wirth*

Nachdem im Kapitel „Grundlagen“ die wichtigen Begriffe für diese Arbeit geklärt und das Problemfeld abgegrenzt wurde, befasst sich dieses Kapitel mit dem Finden einer geeigneten Notation, um modellbasierte Interaktionsanwendungen zu erstellen.

Als Erstes werden hierfür die verschiedenen Arten von Modellen, die bei Interaktionsanwendungen zum Einsatz kommen, untersucht. Problematisch ist dabei die Verbindung zwischen verschiedenen Modellen. Für diese Problemstellung hat sich der Begriff des *Mapping Problems* etabliert, der anschließend erläutert wird.

Damit Modelle als Basis für Anwendungen verwendet werden können, müssen sie in irgendeiner Form festgehalten werden. Dies geschieht mit Hilfe von Notationen. Um eine geeignete Notation zu finden, ist zu untersuchen, welchen Ansprüchen eine Notation genügen muss. Zusätzlich bedarf es eines Schemas, in das die verschiedenen Modelle eingeordnet werden können. Durch die Anforderungen lässt sich auch eine Bewertung vornehmen. Dazu wird ein eigens entwickeltes Schema vorgestellt. Verschiedene repräsentative Notationen werden vorgestellt und bewertet. Das Kapitel endet mit der Wahl einer Notation, die im Rest der Arbeit verwendet wird.

3.1 Modellarten und Mapping Problem

Die Benutzung von Modellen im Bereich interaktiver Benutzerschnittstellen erfüllt verschiedene Zwecke und bietet eine Reihe von Vorteilen [94]. Sie dienen

zur automatischen Generierung von Benutzerschnittstellen, wobei direkt eine Implementierung aus einem Modell generiert wird [185]. Sie unterstützen die Kontrolle der Konsistenz und Vollständigkeit. So lässt sich überprüfen ob z. B. alle vom Benutzer auszuführenden Aktionen über die Navigationsstrukturen erreicht oder aktiviert werden können [93]. Sie ermöglichen die Erzeugung von Hilfe-Systemen, oft kontextabhängig, einschließlich Animationen und Erklärungen [53, 183, 182]. Schließlich sind sie auch beim Evaluieren von erstellten Systemen von Nutzen und ermöglichen z. B. die Messung der Geschwindigkeit, mit der ein Benutzer eine Aufgabe erledigen kann [86].

Angesichts der vielen verschiedenen Zwecke ist es verständlich, dass ein einzelnes Modell oft nicht ausreicht, um alle diese Aufgaben zu erfüllen. Daher kommen eine Reihe von Modellen zum Einsatz, die nun vorgestellt werden.

3.1.1 Arten von Modellen

Modellbasierte Anwendungen verwenden in der Regel mehr als nur ein einzelnes Modell. Es existieren Modelle für verschiedene Aspekte der Anwendung, des Benutzers und noch weitere im Umfeld. Nach Puerta und Eisenstein unterscheidet man die folgenden Modelle [158]:

Das Aufgabenmodell (*user-task model*) beschreibt *Tasks* (Aufgaben), die der Benutzer mit einer Anwendung erledigen will. Es enthält einzelne Elemente, die spezifische Aktionen beschreiben und Elemente, welche Aktionen zu Reihenfolgen und Alternativen zusammenfassen. Zusätzlich können Elemente Informationen über Bedingungen zur Ausführung von Aktionen besitzen.

Das Domänenmodell (*domain model*[1]) definiert Objekte, die im System vorhanden sind und welche Aktionen mit ihnen durchgeführt werden können. Es beschreibt dazu auch die Abhängigkeiten und Relationen von Objekten untereinander sowie die Attribute der Objekte.

Das Benutzermodell (*user model*) beschreibt die verschiedenen Arten von Benutzern, die eine Anwendung nutzen und die Eigenschaften der Benutzer. Es handelt sich hierbei jedoch nicht um ein kognitives Modell, welches Informationsaufnahme und Ähnliches beschreibt, sondern um ein Modell, das Eigenschaften wie Alter, Sehschwächen und Präferenzen von Benutzern beschreibt.

Das Präsentationsmodell (*presentation model*) repräsentiert die verschiedenen Elemente der Benutzerschnittstelle und ihre Ausprägungen (visuell, haptisch usw.). Es enthält auch Informationen zum Stil in denen die Elemente präsentiert werden, z. B. zu Schriftarten oder zu verwendenden Stimmen.

Das Dialogmodell (*dialog model*) legt fest, wie der Benutzer mit dem Präsentationsmodell interagiert. Dazu beschreibt es die möglichen Aktionen des Benutzers und wie die Anwendung darauf reagieren soll.

[1] Auch *concepts model* genannt [25].

Das Umgebungsmodell (*environment model*) wurde von Duarte und Carriço beschrieben [47]. Es liefert kontextuelle Informationen wie beispielsweise über Lautstärke, Helligkeit und den Aufenthaltsort. Die Informationen hängen stark vom Einsatzzweck einer Anwendung ab.

Neben den Modellen von Puerte und Eisenstein existieren noch einige, die in anderen Arbeiten beschrieben wurden. Die für diese Arbeit relevanten Modelle sind das Gerätemodell und das Evaluationsmodell.

Das Gerätemodell (*device model*) beschreibt über Profile die Eigenschaften von eingesetzten Geräten. Einzelne Geräte werden durch ihre Attribute gekennzeichnet. Ein Ausgabegerät besitzt z. B. Attribute für die Auflösung in Pixeln, ein Eingabegerät wie eine Maus verfügt beispielsweise über Attribute für ihre Koordinaten und für die verschiedenen Knöpfe [214].

Das Evaluationsmodell beschreibt wie eine Reihe von Eigenschaften der Benutzerschnittstelle gemessen werden können. Hier kann es je nach den zu messenden Eigenschaften eine Vielzahl von Modellen geben, die eingesetzt werden.

3.1.2 Mapping Problem

Die verschiedenen Modelle stehen nicht für sich allein. Manche Modelle lassen sich im Sinn von Model Driven Software Development zur Codeerzeugung einsetzen. Andere Modelle lassen sich transformieren, so kann z. B. aus einem Präsentationsmodell zusammen mit den Geräteeigenschaften einer Zielplattform ein plattformspezifisches Modell (PSM) generiert werden. Weitere Modelle, wie z. B. manche Dialogmodelle, können zur Laufzeit interpretiert werden.

Modelle können transformiert und miteinander verbunden werden. Die Verknüpfungen werden allgemein als *Mappings* bezeichnet. Durch modellbasierte Erzeugung von Benutzerschnittstellen soll aus den abstrakten Modellen eine konkrete Spezifikation abgeleitet werden.

Als *Mapping Problem* wird die Schwierigkeit der Mappings zwischen verschiedenen Modellen bezeichnet. Mappings können als Verbindung zwischen Modellen, zur Transformation von einem in ein anderes Modell oder auch zur Erzeugung eines neuen Modells durch Kombination aus anderen Modellen eingesetzt werden. Mappings können sehr komplex werden. Auftretende Probleme sollen an einem Beispiel illustriert werden. Für das Fallbeispiel des MP3-Players sei ein Aufgaben-, ein Domänen- und ein Benutzermodell gegeben. Dialog- und Präsentationsmodell sollen aus dem Aufgabenmodell abgeleitet werden. Es ergeben sich Probleme, sobald das Aufgabenmodell geändert wird. So müssen die abgeleiteten Modelle an neue Aktionen des Aufgabenmodells angepasst werden. Unter Umständen ist auch eine Anpassung des Domänenmodells notwendig, falls ein Task mit neuen Objekten arbeitet. Für das Fallbeispiel des MP3-Players könnte z. B. ein neuer Task das zufällige Auswählen des nächsten Lieds sein. Im Dialogmodell muss dazu dann vorgesehen werden, dass der Benutzer diese Funk-

tionalität ein- und ausschalten kann. Im Präsentationsmodell sind für die jeweiligen Modalitäten Anpassungen vorzunehmen, z. B. eine GUI würde um eine entsprechende Taste erweitert werden. Im Domänenmodell muss allerdings sichergestellt werden, dass dieselbe Funktionalität auch vorhanden ist. Eine Playlist-Klasse ist im Beispiel um eine Methode zum Randomisieren ihrer Einträge zu erweitern.

Zunächst sind die verschiedenen Arten von Mappings zu untersuchen. Puerta und Eisenstein beschreiben in ihrer Arbeit sechs Mappings [158].

Task-Dialog Mapping Da Tasks durch Gruppierung zusammengefasst werden gibt es hier verschiedene Möglichkeiten des Mappings. Für einen Ausschnitt aus einem Dialog kann es sinnvoll sein mehrere Tasks zusammenzufassen oder eine Gruppe in mehrere Teildialoge zu trennen.

Task-Presentation Mapping Die Bestandteile einer Benutzerschnittstelle können analog zu Tasks gruppiert werden. Dementsprechend müssen Teile bei einer Umgruppierung im jeweils anderen Modell ebenfalls angepasst werden.

Domain-Presentation Mapping Elemente aus dem Domänenmodell besitzen Attribute, die entweder zur Darstellung wichtig sind oder direkt in ihr angezeigt werden sollen. An dieser Stelle ist es sinnvoll, zusätzlich ein Gerätemodell einzubeziehen. Damit lässt sich das Präsentationsmodell an die Gegebenheiten des Geräts anpassen.

Task-User Mapping Ein Benutzermodell kann verschiedene Arten von Nutzern enthalten, die nur einen Teil der vorhandenen Tasks verwenden. Damit lassen sich auf einfache Art verschiedene Tätigkeiten für Rollen realisieren. Dies führt jedoch zu einer Vervielfachung aller Mappings, die Transformationen sind, da für jede Rolle passende Transformationen vorhanden sein müssen.

Task-Domain Mapping Da durch Tasks der Zustand der Domänenobjekte verändert werden kann, ist dieses Mapping notwendig.

Presentation-Dialog Mapping Die Elemente eines Dialogs müssen mit ihren entsprechenden Präsentationselementen durch ein Mapping verbunden werden.

Zusätzliche Mappings sind möglich, wenn man weitere Kombinationen der Modelle bildet. Diese sollen hier behandelt werden.

Domain-Dialog Mapping Durch Interaktion mit dem Präsentationsmodell ist es möglich, dass schon im Dialogmodell Daten von Domänenobjekten verändert werden sollen. Da zusätzlich Daten von Objekten aus der Domäne den Ablauf des Dialogs beeinflussen können, ist ein Mapping analog zum Task-Domain Mapping notwenig.

Environment-Presentation Mapping Die Eigenschaften einer Umgebung können sich dies stark auf das Präsentationsmodell auswirken. In sehr lauten Umgebungen kann das Präsentationsmodell z. B. die verwendeten Modalitäten der Umgebung anpassen und auf Sprache zu Gunsten von grafischer Ausgabe verzichten.

Umgekehrt verhält es sich bei direkter Sonneneinstrahlung auf Anzeigegeräte, wodurch diese schwer ablesbar werden.

Device-Presentation Mapping Auch die eingesetzten Geräte und ihre Möglichkeiten beeinflussen das Präsentationsmodell. Die Modalitäten, die von Ein- und Ausgabegeräten bereitgestellt werden, lassen sich vom Präsentationsmodell aus verwenden.

User-Presentation Mapping Auch die Eigenschaften des Benutzers beeinflussen das Präsentationsmodell. Ein blinder Benutzer wird keine GUI bedienen, ein Tauber keine Sprachschnittstelle verwenden wollen. Doch so extrem müssen die Beispiele nicht sein, schon Vorwissen, Aufgeregtheit oder eine Sehschwäche sind wertvolle Informationen, um die Präsentation der Benutzerschnittstelle zu optimieren.

Mit passenden Evaluationsmodellen sind zwei weitere Mappings vorstellbar: Das *User-Domain Mapping* und das *User-Dialog Mapping*. Ein User-Domain Mapping findet im Kopf des Benutzers statt. Dieser bildet sich ein mentales Modell, welches mit dem vorhandenen Domänenmodell verglichen werden kann, um Abweichungen und damit potentielle Missverständnisse frühzeitig zu erkennen[2]. Ein User-Dialog Mapping kann verwendet werden, um zu überprüfen, wie gut ein Benutzer der Navigation durch die verschiedenen Tasks folgen kann.

Das *Environment-User Mapping* beschreibt die Einflüsse der Umgebung auf den Benutzer. So können z. B. weitere Menschen in der Nähe eines Benutzers dazu führen, dass dessen Vorlieben für Modalitäten sich ändern. Manche Benutzer stören sich an Benachrichtigungen durch Tonsignale oder Sprache.

Wie durch die Auflistung klar wird, existieren eine ganze Reihe von Mappings. Diese können unterschiedlich komplex sein, da jedes Modell verschieden viele Arten von Elementen enthalten kann. Ein Ziel dieser Arbeit ist es, die Anzahl an Mappings zu reduzieren. Dies kann durch geschickte Wahl von Notationen und die Wahl der richtigen Werkzeuge geschehen. Die Problematik kann dabei nicht umgangen werden, es ist lediglich möglich auf vorhandene Mappings zurückzugreifen und diese wiederzuverwenden. Dabei spielt die Art des Mappings eine Rolle.

3.1.3 Arten von Mappings und Problemen

Wie in den letzten Abschnitten schon anklang, existieren verschiedene Arten von Mappings. Modelle können mit einander verbunden sein und aus einem Modell kann ein neues abgeleitet werden. Auch in der bestehenden Literatur wurde dieses Thema behandelt. Limbourg et al. [98] beschreiben drei Kategorien von Mappings (1–3). Später wurden diese von Clerckx et al. [34] um drei weitere Problempunkte erweitert (4–6).

[2] siehe hierzu auch Don Norman [131]

1. **Model derivation:** Ein vorher nicht spezifiziertes Modell wird aus den Elementen und Relationen eines oder mehrerer bestehender Modelle abgeleitet. Die Parameter der Transformationsregeln müssen dabei festgelegt werden. Zum Beispiel ist es möglich, ein Dialogmodell aus einem Aufgabenmodell abzuleiten, wenn genügend Wissen über die Art der Tasks vorhanden ist.

2. **Model linking/binding:** Die Elemente und Relationen eines oder mehrerer bestehender Modelle werden verarbeitet, um Relationen zwischen Elementen festzustellen, zu aktualisieren oder wieder herzustellen. Dieser Mechanismus kann innerhalb eines Modells zur Bereinigung von Relationen verwendet werden. Er kann genauso gut zwei miteinander verbundene Modelle synchron halten.

3. **Model composition:** Ein oder mehrere vorher spezifizierte Modelle werden teilweise oder komplett vereinigt, um daraus ein neues Modell zu erzeugen oder die ursprünglichen Modelle wieder abzuleiten (wie beim Reverse Engineering). Im zweiten Fall kann man auch von **Model decomposition** sprechen.

4. **Partial model derivation:** Elemente oder Relationen eines Modells werden aus einem anderen Modell abgeleitet. So können z. B. Transitionen zwischen Zuständen eines Dialogmodells aus einem Aufgabenmodell abgeleitet werden.

5. **Model modification:** Das Modell wird durch den Designer verändert, er kann z. B. ein unvollständig generiertes Modell komplettieren.

6. **Model update:** Falls in einem Modell eine Eigenschaft verändert oder hinzugefügt wird, werden in einem verbundenen Modell per Hand vom Designer oder automatisch durch einen Algorithmus, die entsprechenden Eigenschaften aktualisiert.

Das Mapping Problem besteht aus einer Reihe von Teilproblemen. Je nach Art der verwendeten Modelle und der Verbindungen zwischen ihnen treten verschiedene dieser Teilprobleme auf.

3.2 Ansätze zur Klassifikation

Im Folgenden werden verschiedene Ansätze zur Klassifikation von Notationen vorgestellt und ein Verfahren zu Einordnung erarbeitet. Dies soll es ermöglichen, die verschiedenen in Frage kommenden Möglichkeiten zur Beschreibung von Interaktionsanwendungen einzuordnen und zu bewerten. Dabei stellt sich die Frage, in welchem Detailgrad eine sinnvolle Klassifikation und Bewertung vorgenommen werden kann.

An Notationen gibt es eine Reihe von Anforderungen, von denen die relevanten herausgegriffen werden.

Wie im Kapitel Grundlagen bereits vorgestellt, bezeichnet man mit dem Begriff Interaktion nicht die einzelne Handlung eines Benutzers und die Reaktion eines Systems darauf, vielmehr wird der Begriff synonym für Handlungsabläufe verwendet. Darum ist es ein zentrales Anliegen der Arbeit, solche Interaktionsabläufe beschreiben zu können.

Zur Klassifikation muss auch die Abdeckung von Modellen berücksichtigt werden (siehe Abschnitt 3.1.1).

Ein interaktives System besteht aus der Kombination von Inhalt, Form, sowie der Beschreibung des Verhaltens eines Systems durch Anwendungslogik und Interaktionslogik. Darum ist es wichtig die Mappings zwischen den verschiedenen Modellen mit zu betrachten.

Schließlich ist auch die Mächtigkeit der Notation zu bewerten. Bietet sie beispielsweise Konstrukte, um alles Nötige zu beschreiben und sind Abkürzungen für umständliche Schreibweisen vorhanden?

3.2.1 Anforderungen

Zunächst wird mit den allgemeinen Anforderungen, die erfüllt werden sollen, begonnen. Abowd et al. beschrieben bereits 1989 eine Reihe von Anforderungen [2] und andere Autoren wie Weld et al. befassten sich mit Themen wie Personalisierung [205]. Die Anforderungen werden hier gesammelt dargestellt.

Internationalisierung: Eine Benutzerschnittstelle soll sich an verschiedene Sprachen und Kulturen anpassen lassen. Texte sollen in Landessprache vorliegen, Maßeinheiten, Datum, Uhrzeit und viele weitere Elemente sollen in der Konvention eines vorgegebenen Landes ausgegeben werden. Eine Notation sollte Internationalisierung unterstützen oder zumindest nicht einschränken.

Personalisierbarkeit: Eine Benutzerschnittstelle soll an die Bedürfnisse der Endanwender angepasst werden können. In einer GUI können Benutzer beispielsweise Icons in Toolbars austauschen, um oft verwendete Funktionen schnell im Zugriff zu haben. Hier gilt wie bei der Internationalisierung, dass eine Notation diesen Aspekt unterstützen sollte.

Plattformunabhängigkeit: Die Notation soll unabhängig von der eingesetzten Hardware sein. Dies gilt insbesondere in Hinblick auf die verwendeten Modalitäten.

Erweiterbarkeit um Modalitäten: Es soll im Nachhinein möglich sein, die Benutzerschnittstelle um weitere Modalitäten zu erweitern.

Redundanz: Die Notation soll mit möglichst wenig Redundanz arbeiten.

Beschreibung von Struktur: Strukturierende Elemente sind in der Notation vorzusehen.

Universalität: Die Notation soll für ein möglichst breites Spektrum von Anwendungen einsetzbar sein.

Konsistenz: Die Interaktion soll immer gleich verlaufen. Eine Funktion des Systems sollte immer gleich anzusprechen sein, wie Dix et al. beschreiben [46, S. 584]. In einer GUI sollen deshalb Elemente vorhersehbar an den gleichen Stellen platziert werden wie bei der letzten Verwendung einer Anwendung.

Ergonomie: Die Benutzerschnittstellen sollen ergonomischen Anforderungen genügen, wie sie z. B. von der DIN EN ISO 9241-110 aufgestellt wurden [77].

Eine weitere wichtige Eigenschaft ist die **Erweiterbarkeit der Notation**. Richter nennt sie in seiner Arbeit *Dynamische Erweiterbarkeit* [166]. Bei Richter sind es Elemente von Toolkits, in dieser Arbeit ist der Begriff weiter gefasst, um auch die Erweiterbarkeit einer Notation um neue Notationselemente abzudecken.

Van den Bergh und Coninx [82] beschrieben 2004 eine Reihe von weniger formalen Anforderungen an das Arbeitsumfeld.

Verständlichkeit: Die Notation soll von möglichst vielen Leuten einfach verstanden werden. Verständnis lässt sich jedoch schwer quantifizieren.

Angemessenheit der Notation: Grafische Notationen sollen für verschiedene Aspekte der Spezifikation unterschiedliche Arten von Diagrammen nutzen. Diese sollen ihrem Einsatzzweck angemessen sein. Auch textuelle Notationen können größere Klarheit erreichen, wenn Aspekte eines Modells getrennt betrachtet werden.

Keine isolierten Teilmodelle: Es soll ein gesamtes Modell beschrieben werden. Falls verschiedene Teilmodelle beschrieben werden, stellen Relationen zwischen den Teilen die Verbindung her.

Ausdrucksstärke: Zusätzlich zur Anforderung der Verständlichkeit soll eine Notation nicht langatmig werden. Sie muss es ermöglichen, komplexe Zusammenhänge oder Modelle zu beschreiben.

Werkzeugunterstützung: Die Notation soll eine einfache Anwendung und Umsetzung des Interaktionsdesigns ermöglichen. Hierfür sollen Werkzeuge für die Modellierung und die Umsetzung in verschiedenen Modalitäten zur Verfügung stehen.

Abowd et al. beschrieben in ihrer Arbeit von 1989 auch Trade-offs [2].

Analyse vs. Ausdrucksstärke: Eine sehr formale Notation kann die Analyse eines Dialogs erschweren, da komplexe Parameter zu berücksichtigen sind. Sie ist aber wichtig, um automatisch Benutzerschnittstellen zu erzeugen.

Spezifikation vs. Ausführung: Eine Notation, die zur Spezifikation eines Modells geeignet ist, kann für die Ausführung des Modells ungeeignet sein.

Ein Punkt der von keinem der genannten Autoren aufgeführt wird ist die **Verbreitung** und damit indirekt die Akzeptanz einer Notation. Wird eine Notation nur von einer kleinen Gruppe eingesetzt, so bestehen größere Hürden sie zu nutzen, als wenn es sich um eine gebräuchliche Notation handelt.

Zusammenfassung der Eigenschaften

Für diese Arbeit sind einige der aufgezählten Eigenschaften vernachlässigbar. Nicht jede Notation muss erweiterbar sein, dies ist nur notwendig, wenn die vorhandenen Elemente nicht für alle Einsatzzwecke ausreichen. Einige grundsätzliche Eigenschaften müssen jedoch bei jeder zu bewertenden Notation vorhanden sein. Jede Notation verfügt über Möglichkeiten zur Strukturierung ihrer Elemente. Redundanz ist in fast jeder Notation vorhanden und wird nur dann zu einem Problem, falls sie Überhand nimmt. Die Notationen werden so ausgewählt, dass sie Internationalisierung und Personalisierbarkeit unterstützen oder zumindest nicht einschränken. Weiter erfolgte bei der Auswahl der Notationen eine Beschränkung auf plattformunabhängige Notationen. Alle vorgestellten Notationen sind weiter so aufgebaut, dass ihre Teilmodelle mit einander verbunden sind.

Über Abwägungen wie die Analysestärke gegen die Ausdruckskraft und Einsatz zur Spezifikation gegen Ausführung der Notation kann nur berichtet werden, da diese Entscheidungen von den Schöpfern einer Notation getroffen werden.

Keinen Einfluß auf die Wahl einer Notation soll die Ergonomie von Benutzerschnittstellen haben. Notationen sollen soweit von Benutzerschnittstellen entkoppelt sein, dass sie nur einen indirekten Einfluß auf diese Eigenschaft besitzen. Ebenso soll die Konsistenz von Interaktionsabläufen keine zu große Rolle spielen. Eine Notation soll die Interaktion einheitlich beschreiben. Inkonsistenzen können dann nur bei der Ableitung verschiedener Benutzerschnittstellen aus einem abstrakten Modell entstehen. Diese Inkonsistenzen zu beseitigen ist allerdings Schwerpunkt von Arbeiten, die sich mit der Generierung von Benutzerschnittstellen befassen.

Eine Werkzeugunterstützung ist nicht zwingend erforderlich. Vorhandene Werkzeuge sind jedoch prinzipiell von Vorteil, da sie die Anwendung der Notation fördern.

Schließlich soll eine Notation auch möglichst verbreitet und damit von den an der Anwendungserstellung beteiligten Gruppen akzeptiert sein.

Es ergeben sich damit sieben Punkte, die für alle Notationen abzuprüfen sind.

1. Erweiterbarkeit um Modalitäten
2. Angemessenheit der Notation
3. Verständlichkeit

4. Ausdrucksstärke
5. Universalität
6. Werkzeugunterstützung
7. Verbreitung

3.2.2 Frühe Aufgliederung nach Abowd et al.

In ihrer Studie über Sprachen zur Beschreibung von Benutzerschnittstellen von 1989 stellten Abowd et al. eine Dreiteilung von interaktiven Anwendungen in Benutzer, System und Interaktion vor [2, S. 31].

Für den Benutzer wird auf ein Modell abgezielt, welches beschreibt, was im Benutzer vorgeht und über welche Metaphern, internen Modelle und Ziele er verfügt. Dieses Modell deckt sich mit dem oben vorgestellten Begriff des Benutzermodells.

Unter System verstehen Abowd et al. den Aufbau von Anwendungen. Darunter fallen Modelle der Systementwickler und Aussagen über die Objekte aus der Anwendungsdomäne. Abowd et al. arbeiteten an frühen Spezifikationstechniken und Sprachen zum Prototyping. Dieser Bereich lässt sich in der Terminologie dieser Arbeit jedoch gut beim Domänenmodell ansiedeln.

Unter Interaktion wird der Ablauf verstanden, in dem der Austausch von Informationen zwischen Benutzer und Anwendung geschieht. Der Bereich deckt sich mit dem Dialogmodell.

In diesem sehr frühen Ansatz fehlen mehrere Modelle, es werden keine Aussagen zur Präsentation der Benutzerschnittstelle gemacht, Ansätze zu einem Aufgabenmodell sind noch nicht vom Dialogmodell getrennt und über die Umgebung und die verwendbaren Geräte können ebenfalls keine Aussagen getroffen werden.

Es wird zwar eine grobe Strukturierung vorgeschlagen, doch keine spezifischen Notationen eingesetzt, da es sich nur um einen konzeptionellen Ansatz handelt. Eine Erweiterung des Systems bleibt aber möglich. Der Ansatz ist universell genug um auf beliebige Anwendungen übertragbar zu bleiben.

Nach dieser Veröffentlichung von Abowd et al. gab es bis zur Veröffentlichung des Cameleon Frameworks, das im nächsten Abschnitt vorgestellt wird, keine Arbeiten, die dediziert Modelle und ihre Zusammenhänge betrachtet.

3.2.3 Cameleon Framework

2003 stellten Calvary et al. ein Framework für Benutzerschnittstellen mit vielen Zielplattformen vor [26], welches im Rahmen des Cameleon Forschungsprojekts

entwickelt wurde. Vanderdonckt et al. nutzten dieses Framework als Basis für USIXML, einer Notation, die im Anschluß vorgestellt wird und den Kern des Frameworks kompakt präsentiert [196].

Das Framework gibt vor, dass eine Notation für Benutzerschnittstellen eine Beschreibung auf verschiedenen Ebenen vornehmen muss. Jede Ebene ist dabei auf eine Aufgabe spezialisiert, wie in Abbildung 3.1 illustriert.

Die Ebene *Tasks and Concepts* beschreibt die auszuführenden Vorgänge und die domänen-spezifischen Konzepte, die für die Vorgänge nötig sind.

Die Ebene des *Abstract User Interface* bestimmt den Raum der Interaktion. Teilaufgaben werden hier zusammengefasst, Patterns verwendet und Analysen der benötigten kognitiven Kapazitäten lassen sich durchführen. Das UI ist noch unabhängig von Modalitäten.

Auf der Ebene des *Concrete User Interface* werden konkrete Interaktionsobjekte ausgewählt. Interaktionsobjekte können z. B. Elemente aus Toolkits sein. Diese Ebene ist noch unabhängig von der endgültig verwendeten Plattform.

Die Ebene des *Final User Interface* beschreibt das ausführbare User Interface. Das FUI wird entweder interpretiert wie z. B. eine Webseite oder besteht aus binärem Code.

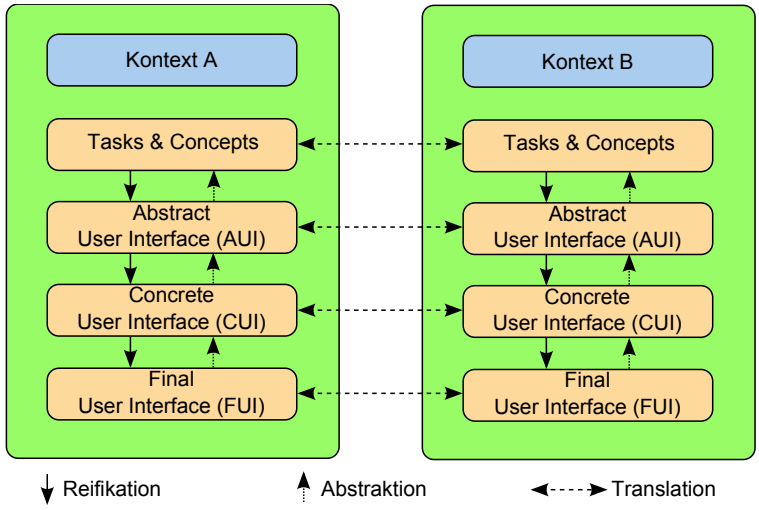


Abbildung 3.1: Modelltransformationen im Cameleon Framework.

Das Cameleon Framework beschreibt Transformationen zwischen den verschiedenen Ebenen. Es ist möglich von einer Ebene zu einer höheren zu abstrahieren

(Abstraktion) oder zu einer tieferen zu konkretisieren (Reifikation). Innerhalb einer Ebene kann zwischen zwei Kontexten übersetzt werden. So kann beispielsweise zwischen einem FUI in HTML und WML in beide Richtungen eine Übersetzung (Translation) stattfinden. Die Translation zwischen verschiedenen Kontexten wurden bisher nur von Calvary et al. beschrieben [26].

Das Cameleon Framework ist dabei unabhängig vom Entwicklungsansatz. Es kann klassisch im Wasserfallmodell entwickelt werden, wobei nur Reifikationen stattfinden, aber auch Reverse Engineering wird durch die Abstraktion vorgesehen. Es hat sich gezeigt, dass der schrittweise Übergang von abstrakten zu konkreteren Modellen im Lauf des Entwicklungsprozesses eine zu frühe Festlegung auf einzelne Elemente der endgültigen Benutzerschnittstelle vermeidet, wie da Silva und Barclay beispielsweise mit dem Teallach-System festgestellt haben [179, 11].

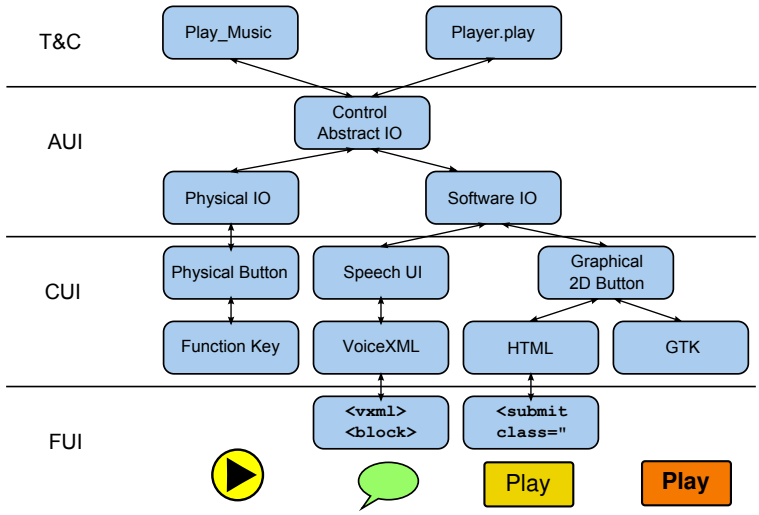


Abbildung 3.2: Transformationen und entsprechende Elemente am Beispiel des MP3-Players.

Am folgenden Beispiel sollen die verschiedenen Ebenen des Cameleon Frameworks noch genauer analysiert werden. In Abbildung 3.2 sieht man die Ebenen beim Abspielen eines Liedes aus dem zweiten Fallbeispiel. Auf der Ebene *Tasks and Concepts* existiert dazu ein Objekt `Player` und ein Task `Play_Music` zum Abspielen von Musik. Dem Objekt werden auf der abstrakten Ebene Eingabe-Interaktoren für Hardware und Software zugeordnet. Auf der CUI-Ebene entsprechen diese Tasten, Buttons oder Sprachbefehlen. Die eigentliche Schnittstelle wird bei der Ausführung durch Binärcode oder die Interpretation von Markup-

Sprachen übernommen. Die Interaktion eines Benutzers wird durch die verschiedenen Ebenen übertragen. So kann auf die Funktionalitäten der Domänenobjekte zugegriffen werden. Ein Benutzer kann z. B. die Methode `play` des Objekts `Player` aufrufen, welche mit der Wiedergabe der Musik beginnt.

Das Cameleon Framework zeigt, wie Modelle zur Entwicklung interaktiver Anwendungen nach den Prinzipien des Model Driven Software Development eingesetzt werden und welche Transformationen dabei zum Einsatz kommen. Insbesondere beschreibt es die Translation zwischen verschiedenen Modellen.

Von Vorteil ist, dass das Framework universell einsetzbar ist und keine Notationen vorschreibt.

Ein Nachteil ist, dass das Aufgabenmodell und das Domänenmodell zusammengefasst werden. Es handelt sich um zwei verschiedene Modelle, die so den Eindruck erwecken, ein einzelnes Modell zu sein.

Das Framework macht keine Aussagen über Internationalisierung oder Personalisierung. Diese sind allerdings in der Ebene der Final User Interfaces anzusiedeln.

3.2.4 Fazit

Neben der frühen Arbeit von Abowd et al., der Arbeit Puertas und Eisensteins (siehe Abschnitt 3.1.2 zum Mapping Problem) und dem Cameleon Framework gibt es nach meinem Kenntnisstand keine Arbeiten, die einen systematischen Überblick über alle verwendbaren Modelle und den Mappings zwischen ihnen bieten.

Bisherige Klassifikationen decken den zu modellierenden Raum nicht vollständig ab. Zusätzlich zu den oben ausgewählten Anforderungen ist ein erweitertes Klassifikationsframework notwendig, welches alle relevanten Modelle und Mappings beinhaltet.

3.3 Models and Mappings-Klassifikationsframework

Um eine umfassende Klassifikation des Einsatzes von Modellen für Benutzerschnittstellen vornehmen zu können, ist auch das Cameleon Framework nicht ausreichend. Das Cameleon Framework konzentriert sich auf die Transformationen zur Abstraktion, Reifikation und Translation zwischen Modellen. Es fehlen Möglichkeiten, eine Reihe von Teilmodellen detaillierter zu beschreiben.

Im Folgenden wird das *Models and Mappings*-Klassifikationsframework vorgestellt, das für diese Arbeit entwickelt wurde. Es ist zusätzlich um ein Gerätemodell, ein Benutzermodell und ein Umgebungsmodell erweitert. Ein Evaluationsmodell wird für die Klassifikation vorgesehen, ist aber für die Erstellung von interaktiven Anwendungen nicht direkt notwendig.

Zusätzlich zu den Modellen muss eine Klassifikation Aussagen über die relevanten Mappings zwischen den Modellen treffen können. Es spielt eine große Rolle, ob es sich bei einem Mapping um eine Verknüpfung oder eine Transformation von Modellen handelt. Hierzu werden die verschiedenen Arten von Mappings aus Abschnitt 3.1.3 herangezogen.

In Abbildung 3.3 sieht man das Models and Mappings Klassifikationsframework. In der Abbildung sind für alle verwendbaren Modelle Bereiche vorgesehen. *Task* steht für das Aufgabenmodell, *Domain* für das Domänenmodell, *Dialog* für das Dialogmodell, *Presentation* für das Präsentationsmodell, *Device* bezeichnet das Gerätemodell, *Environment* das Umgebungsmodell, *User* das Benutzermodell und *Evaluation* das Evaluationsmodell.

Die Modelle sind entsprechend der möglichen Verknüpfungen angeordnet. Task-, Dialog- und Präsentationsmodell können auf Objekte des Domänenmodells zugreifen. Das Präsentationsmodell ist mit Benutzer-, Geräte- und Umgebungsmodell verbunden, um deren Einflüsse nutzen zu können.

Ein nicht verwendetes Modell wird im Diagramm dunkelgrau eingefärbt. In der Abbildung werden kein Evaluations- und kein Umgebungsmodell eingesetzt.

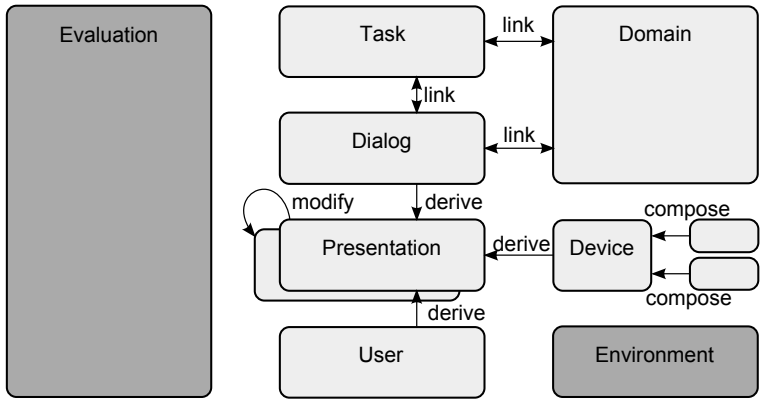


Abbildung 3.3: Das Models and Mappings-Klassifikationsframework.

Die verschiedenen Modelle werden durch Pfeile verbunden, entlang derer die verwendeten Mappings vermerkt werden. Wenn zwischen Modellen kein Mapping existiert wird kein Pfeil eingezeichnet. Für die in Abschnitt 3.1.3 beschriebenen Mappings werden die folgenden vier Kürzel verwendet:

1. **derive** für das Ableiten aus einem Modell,
2. **link** für Verknüpfungen zwischen Modellen,

3. **compose** für das Zusammensetzen von Modellen zu einem neuen Modell,
4. **modify** für die Modifikation eines Modells durch einen Entwickler.

Geschieht eine teilweise Ableitung eines Modells, wird dies im beschreibenden Text zum Diagramm vermerkt. Auch ein *Model update* wird nicht im Diagramm markiert, da es Folgen an vielen Stellen der beteiligten Modelle haben kann und somit nicht zur Übersichtlichkeit beiträgt. Im Erläuterungstext zur jeweiligen Abbildung wird beschrieben, welche Modelle durch die Mappings zusammenwirken.

In Abbildung 3.3 sieht man drei *deriv* Verbindungen zum Präsentationsmodell, welches aus dem Dialogmodell unter Einfluss des Benutzer- und des Gerätemodells abgeleitet wird. Das Domänenmodell ist mit dem Aufgaben- und dem Dialogmodell verbunden, eine weitere Verbindung besteht zwischen Aufgaben- und Dialogmodell. Das Gerätemodell setzt sich aus zwei Teilmodellen zusammen, dies wird durch die beiden Pfeile mit der Beschriftung *comp* verdeutlicht. Beim Präsentationsmodell ist ein *mod*-Mapping vorhanden. Dies bedeutet, dass eine Veränderung einzelner Attribute am Modell vorgenommen wurde. Soll das Präsentationsmodell erneut abgeleitet werden, so müssen diese Veränderungen auch beim neu abgeleiteten Modell vorgenommen werden.

Das vorgestellte Klassifikationsframework bietet mehrere Vorteile. Das Aufgabenmodell und Domänenmodell sind im Gegensatz zum Cameleon Framework klar von einander getrennt, da das Aufgabenmodell Verhalten und das Domänenmodell Anwendungseigenschaften beschreibt. Es sind alle relevanten Modelle im Framework enthalten und auch Mappings zwischen den Modellen sind einfach darstellbar.

Im Vergleich zum Cameleon Framework wird noch ein weiterer Vorteil sichtbar. Im Cameleon Framework ist eine Auffächerung in verschiedene Benutzerschnittstellen möglich, wie Abbildung 3.2 illustriert. Das neue Framework bleibt in diesem Punkt kompatibel, da das Concrete User Interface und das Final User Interface beide im Präsentationsmodell enthalten sind. Es bleibt den Entwicklern überlassen, innerhalb des Präsentationsmodells die Granularität zu wählen. So genügen z. B. vielen Entwicklern Toolkits zur Umsetzung des Präsentationsmodells, diese entsprechen einer Modellierung auf Ebene eines CUIs. Bei Bedarf bleibt es jederzeit möglich eine Reifikation im Präsentationsmodell auf die Ebene von FUIs vorzunehmen.

Bewertung bisheriger Ansätze

Im Rest dieses Kapitels werden bisherige Ansätze zur Modellierung von Interaktionsanwendungen bewertet. Diese lassen sich in mehrere Gruppen gliedern. Eine frühe Unterteilung stammt von 1995 von Myers, der eine Reihe weiterer Kategorien unterschied [119].

Die verschiedenen Ansätze zur Beschreibung von Benutzerschnittstellen werden in zwei Kategorien mit Untergruppen eingeteilt. Die erste Kategorie sind Ansätze, die Dialoge und Oberflächen bzw. deren Verhalten beschreiben. Hierzu gehören Grammatiken, Ereignissprachen, Produktionsregeln und Zustandsdiagramme. Die zweite Kategorie machen Ansätze aus, die abstrakte Modelle verwenden; die wichtigsten Repräsentanten werden vorgestellt. Einige weitere Kategorien sind für diese Arbeit nicht relevant, wie z. B. Screen Scraper, die eine grafische Oberfläche für alte Anwendungen ohne eine solche bieten.

Die Ansätze werden nach den Anforderungen, die in diesem Kapitel gesammelt wurden, bewertet. Zusätzlich werden sie mit dem entwickelten Klassifikationsframework auf die Abdeckung an Modellen und Mappings hin untersucht.

Nach der Bewertung der verschiedenen Ansätze werden noch spezialisierte Modelle und Sprachen für die übrigen Modelle des Klassifikationsframeworks vorgestellt. Diese Sprachen und Modelle vereinfachen die Umsetzung von Systemen. Es finden sich Modelle zur Beschreibung von Benutzereigenschaften, von Geräten und von einzelnen Modalitäten.

3.4 Ansätze zur Beschreibung von Dialogen und Oberflächen

Die erste Kategorie von Ansätzen beschreibt Dialoge, Oberflächen und deren Verhalten. Sie besteht aus frühen, etablierten Ansätzen. Zum Teil wurden sie für andere Zwecke entwickelt, ließen sich dann jedoch auch auf Benutzerschnittstellen übertragen. Es werden drei Gruppen von Notationen vorgestellt: Grammatiken, Ereignissprachen und die mit ihnen verwandten Produktionssysteme sowie Zustandsdiagramme. Aus jeder Gruppe wird ein Repräsentant mit einem Beispiel ausführlicher illustriert.

3.4.1 Grammatiken

Grammatiken wurden zur Spezifikation von GUIs verwendet. Reisner verwendet in seiner Arbeit von 1981 die Backus-Naur Form (BNF) um einen Dialog zu beschreiben [164]. BNF dient meist als formale Notation zur Spezifikation von Programmiersprachen. Donald Knuth schlug den Namen 1964 in einem Leserbrief vor und er wurde allgemein übernommen [92].

Die Elemente der BNF sind Sequenz, Auswahl und Rekursion. Die Grammatik wird durch Regeln festgelegt. Durch das Konzept der Rekursion lässt sich auch eine Iteration darstellen. Nebenläufige Dialoge lassen sich nicht in BNF darstellen.

Recht ähnlich zur BNF sind reguläre Ausdrücke. Diese werden häufig in Editoren oder Skripten verwendet um in Texten zu suchen oder lexikalische Analysen

vorzunehmen. Die Notation geht auf Stephen C. Kleene zurück [90], der sie reguläre Mengen nannte. Durch Ken Thompson, der diese Notation in einem Editor einsetzte, wurden reguläre Ausdrücke bekannt [187].

Die Notation schwankt je nach Werkzeug, die Grundkonzepte sind jedoch immer gleich: Sequenz und Iteration. Durch das Fehlen von Rekursion sind reguläre Ausdrücke nicht so ausdrucksstark wie die Backus-Naur Form.

Die Gruppe der Grammatiken wird am Fallbeispiel des MP3-Players illustriert, dessen Benutzerschnittstelle in BNF beschrieben wird.

```
1 <player>   ::= <volume> | <playlist> | <controls>
2 <volume>   ::= inc_volume | dec_volume
3 <playlist> ::= previous | next
4 <controls> ::= play | pause | stop
```

Listing 3.1: Dialogmodell des MP3-Players in BNF.

In Listing 3.1 sieht man Nichtterminale, die durch <> markiert werden und Terminale. Die ::= trennen die Seiten der Regeln. Durch das Symbol für Oder (|) ist die Auswahl zwischen Alternativen möglich.

Grammatiken bieten eine Reihe von Vorteilen. Sie sind erweiterbar und je nach Design sehr ausdrucksstark. Sie können universell gestaltet werden. Dem gegenüber steht die Gefahr, eine unverständliche oder nicht allgemein genug gehaltene Grammatik zu erstellen. Wie man am Beispiel sieht, bleibt die Beschreibung des Dialogs kompakt. Ein Nachteil ist, dass Nebenläufigkeit nicht direkt unterstützt wird. Im Beispiel kann nie gleichzeitig die Lautstärke geändert werden, während eine andere Aktion ausgeführt wird. Auch die Berücksichtigung des unterschiedlichen Verhaltens der Aktionen ist nicht einfach möglich. Aus dem Pause-Modus wird auch durch `stop` das Lied angehalten, aber wählt man danach `play`, so startet das Lied vom Anfang im Gegensatz zur pausierten Stelle. Dieses Verhalten geht aus der Grammatik allerdings nicht hervor.

Im Klassifikationsframework lässt sich durch eine Grammatik meist nur ein Modell abdecken. Es ist zwar möglich, eine Grammatik zu finden, in der z. B. ein Domänenmodell mit einer Programmiersprache umgesetzt werden kann und das gleichzeitig Elemente für ein Aufgaben- und ein Dialogmodell enthält, solche Grammatiken existieren jedoch nicht für den praktischen Gebrauch. Grammatiken werden von technischen Systemen in hohem Umfang eingesetzt, allerdings gibt es nur eine geringe Zahl geschulter Anwender, die Grammatiken erstellen.

3.4.2 Ereignissprachen und Produktionssysteme

Ereignissprachen und Produktionssysteme repräsentieren beide regelbasierte Ansätze. In solchen Systemen existiert eine Menge von Regeln, die festlegen, wie

auf bestimmte Ereignisse (*Events*) reagiert werden soll. Eine Regel kann mehrere neue Ereignisse auslösen, auf welche dann die selbe oder andere Regeln anwendbar sind. Alle Regeln sind parallel aktiv, eine Notwendigkeit, die sich aus der Beschreibung von verteilten Prozessen und ähnlichen Problemstellungen ergab. Sowohl Ereignissprachen als auch Produktionssysteme lassen sich zur Beschreibung von Dialogmodellen einsetzen. Der Unterschied zwischen ihnen betrifft ihre Einsatzgebiete, so dass beide in diesem Abschnitt behandelt werden. Die wichtigsten Repräsentanten sind *Communicating Sequential Processes* (CSP) und *Production Rules*, die im Folgenden vorgestellt werden.

Communicating Sequential Processes wurden 1978 von Hoare vorgestellt [65, 66], um sequentielle und parallele Prozesse beschreiben zu können. Die Notation wird seit geraumer Zeit auch zur Beschreibung von Benutzerschnittstellen eingesetzt. Als Weiterentwicklungen existieren *Specifying and Prototyping Interaction* (SPI) von Heather Alexander [5] und *Agents* von Gregory Abowd [3].

CSP legt mit wenigen Sprachelementen Regeln fest. Eine Regel besteht aus zwei Seiten, die durch ein Gleichheitszeichen verbunden sind. Operatoren ermöglichen die Beschreibung paralleler Abläufe, Sequenzen, Auswahl zwischen verschiedenen Möglichkeiten sowie Ein- und Ausgabeereignissen. Es ist auch möglich, Regeln zu schreiben, die Parameter entgegen nehmen. Damit wird es einfacher, z. B. Mehrfenstersysteme zu beschreiben, die von der Form her identisch sind und sich nur in Inhalten wie Titel und Text eines Dialogfensters unterscheiden.

Produktionsregeln wurden von Olsen 1990 vorgestellt [41]. Sie haben die Form $Event : Condition \rightarrow Action$. Auch hier sind alle Regeln gleichzeitig aktiv. Ein Ereignis löst die Bearbeitung einer Regel aus. Sobald die Bedingung der Regel erfüllt ist, wird die dazugehörige Aktion ausgeführt. Aktionen können Ausgaben an den Benutzer eines Systems oder Änderungen des internen Zustands des Systems sein. Ausgabeereignisse werden dem Benutzer mitgeteilt und nicht weiter verarbeitet. *Event, Condition, Action* (ECA) fand durch den Einsatz in regelbasierten Systemen eine Verbreitung und wird von weiteren Ansätzen verwendet.

Listing 3.2 zeigt Produktionsregeln am Beispiel des MP3-Players. Im oberen Teil sind Attribute und ihre möglichen Zustände definiert, im unteren Teil die Regeln, die die Änderungen des Systemzustands zur Laufzeit beschreiben. Dabei werden Nutzerereignisse mit Großbuchstaben begonnen, interne Ereignisse mit Kleinbuchstaben geschrieben und um Ausgabeereignisse werden Klammern gesetzt.

Im Beispiel sieht man zwei Bedingungen für `Play`. Falls der Player gestoppt war beginnt das Lied neu, bei einer Pause wird an der vorherigen Stelle fortgefahren. Für Lautstärke und Playlist sind Zustände modelliert, der Einfachheit halber wurden jedoch keine Regeln mit Bedingungen für alle Zustände der Lautstärke und Playlist geschrieben.

```
# Attribute
Volume: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Song:   {s0, s1, s2, s3, s4, s5}
State:  {playing, paused, stopped}

# Regeln
Inc_volume     -> vol-inc
Dec_volume     -> vol-dec
Previous       -> song-dec
Next           -> song-inc
Play:  stopped -> playing player-play (display-play)
Play:  paused  -> playing player-unpause (display-play)
Pause          -> paused player-pause (display-pause)
Stop           -> stopped player-stop (display-stop)
```

Listing 3.2: Der MP3-Player mit Produktionsregeln nach Olsen [41].

Ereignissprachen und Produktionsregeln bieten mehrere Vorteile. Sie sind erweiterbar, universell einsetzbar und die Regeln sind ausdrucksstark. Die Verständlichkeit der Beschreibung einer Anwendung hängt von der Gestaltung der Regeln ab. Wie man im Beispiel sieht ist es möglich Aufrufe von Teilen des Domänenmodells in die Notation einzubetten (z. B. `player-unpause`). Die Notation macht die Trennung zwischen Dialog- und Domänenmodell nicht klar, denn direkt nach einem Zustand wie `playing` steht eine Operation auf einem Domänenobjekt in der gleichen Schreibweise. Vorteilhaft ist dafür, dass durch die Klammerung von Ausgabeereignissen eine saubere Schnittstelle zum Präsentationsmodell vorhanden ist. Im Klassifikationsframework werden bei dieser Notation zusätzlich zum Dialogmodell die Verbindungen zu Präsentations- und Domänenmodell gekennzeichnet. Nach ihrem Höhepunkt in den 1980er und 1990ern hat die Verbreitung solcher Systeme zu Gunsten anderer Verfahren nachgelassen.

3.4.3 Zustandsdiagramme

Zustandsdiagramme (engl. *state charts* oder *state transition networks*) existieren schon seit den 1950ern. Sie sind gerichtete Graphen, in denen Knoten Zustände markieren und Kanten Übergänge bezeichnen [105, 113].

Harel erweiterte die Zustandsdiagramme 1987 um Hierarchie und Nebenläufigkeit, um dadurch die Anzahl von Knoten im Zustandsraum reduzieren zu können [58, 59]. Abbildung 3.4 zeigt einen Teil eines Zustandsdiagramms einer Uhr. Durch eine gestrichelte Linie wird angezeigt, dass nebenläufige Zustände existieren. In der Abbildung ist die Zeitmessung nebenläufig zur Anzeige der Uhrzeit.

Bereits 1968 stellte Newman ein System vor, bei dem das Verhalten einer GUI durch austauschbare Zustandsdiagramme beschrieben wird [124]. Auch heu-

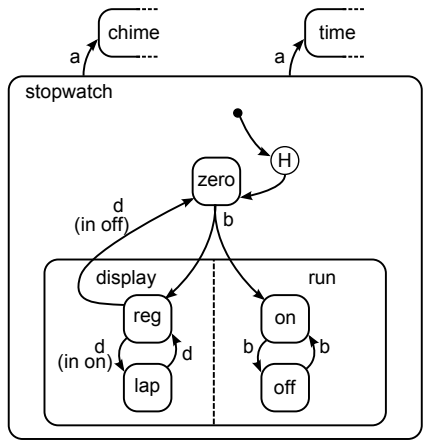


Abbildung 3.4: Ausschnitt aus dem Zustandsdiagramm einer Uhr nach der ursprünglichen Arbeit Harels von 1987.

te werden reine Zustandsdiagramme zur Beschreibung von Interaktion eingesetzt, so z. B. von Hinckley et al. zur Beschreibung von zweihändiger Eingabe am Desktop [64]. Für das Fallbeispiel des MP3-Players lässt sich ein Zustandsdiagramm für das Dialogmodell verwenden.

In Abbildung 3.5 wird das resultierende Dialogmodell dargestellt. Von Vorteil ist die Strukturierung des Diagramms. Zustandsdiagramme sind zwar nicht erweiterbar, doch lassen sie sich universell einsetzen um abstrakt Dialoge oder Tasks zu beschreiben. Sie sind intuitiv verständlich und ausdrucksstark. Ein weiterer Vorteil ist die Verfügbarkeit von Werkzeugunterstützung und ihre weite Verbreitung.

Im Klassifikationsframework finden sie im Aufgaben- und Dialogmodell Einsatz. Verbindungen zum Präsentations- und Domänenmodell sind einfach möglich, da an den Übergängen zwischen den Zuständen Ausdrücke der Art *Event [Condition]/Action* stehen können. Durch die *Actions* lassen sich sowohl Elemente des Präsentations- als auch des Domänenmodells ansprechen.

3.4.4 Fazit

Die vorgestellten Ansätze zur Beschreibung von Dialogen haben ihre jeweiligen Stärken. Grammatiken sind erweiterbar und können sehr universell gestaltet werden. Ereignissprachen und Produktionsregeln sind ausdrucksstark und bieten eine erste Verknüpfung von Dialogmodell und Domäne. Zustandsdiagramme

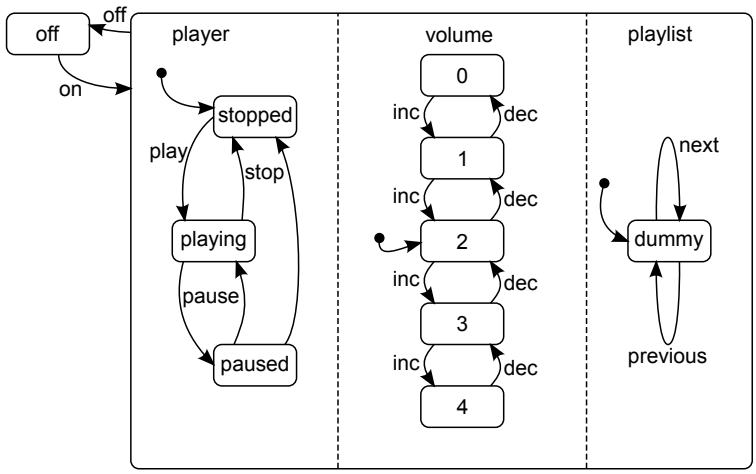


Abbildung 3.5: Dialogmodell des MP3-Players als Zustandsdiagramm.

bieten eine gute Übersichtlichkeit durch ihre Strukturierungsmöglichkeiten und sie bieten die Vorzüge der Event Condition Action-Regeln an ihren Übergängen.

Alle Ansätze beschränken sich jedoch sehr stark auf das Aufgaben- und Dialogmodell. Sie bieten eine sehr einfache Verknüpfungen zu Domänen- und Präsentationsmodell. Abbildung 3.6 zeigt die Einordnung von Harels Zustandsdiagrammen in das Klassifikationsframework.

Abgesehen von den bereits vorgestellten Notationen existieren neuere Ansätze, die ebenfalls textuell verschiedene Modelle beschreiben. An dieser Stelle sei auf Trewin et al. verwiesen [190], die eine Auswertung der vier bekanntesten Ansätze vorgenommen haben: XForms, UIML, XIML und AIAP. Alle diese Ansätze bieten Fortschritte in verschiedenen Bereichen, unterliegen allerdings noch Einschränkunen. XForms besitzt ein Dialogmodell, verbindet aber nicht die einzelnen Dialoge untereinander [215]. UIML beschränkt sich auf Dialoge und ein Präsentationsmodell, das nur eine Plattform unterstützt. Für weitere Plattformen muss die Beschreibung des Modells kopiert und angepasst werden [155, 60]. XIML ist eine Metasprache, in der lediglich die Beschreibung von Elementen für verschiedene Modelle möglich ist, ein standardisiertes Vokabular ist jedoch nicht vorhanden [159, 160]. AIAP wurde zur Erzeugung von Benutzerschnittstellen für mobile Geräte in einer Dienste-Architektur entwickelt und legt seinen Schwerpunkt auf die Verbindung eines Domänenmodells mit einem Dialogmodell [216][3]. Diese Notationen beschränken sich alle auf einen sehr kleinen Be-

[3] Mittlerweile ist AIAP ein INCITS Standard: [74, 75, 71, 73, 72].

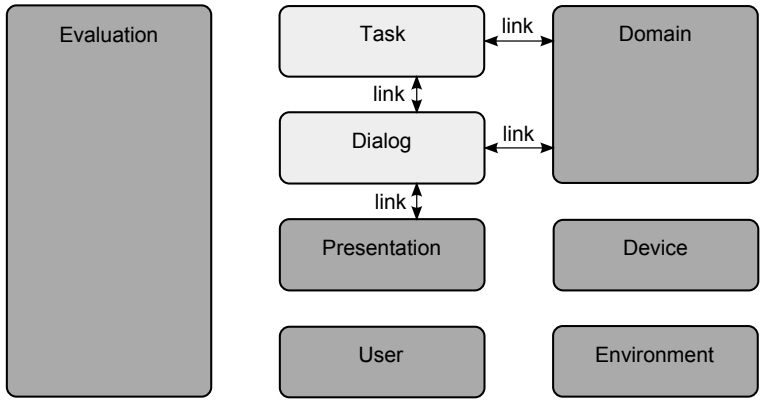


Abbildung 3.6: Einordnung von Zustandsdiagrammen in das Klassifikationsframework.

reich im Klassifikationsdiagramm. Allein mit ihnen lassen sich keine modellbasierten Interaktionsanwendungen erstellen. Wünschenswert wären Ansätze, die mehr Modelle zugleich abdecken können.

3.5 Abstrakte Modelle für Interaktion

Zentral für Interaktionsanwendungen ist eine Abdeckung des Aufgaben- und Domänenmodells sowie die Verbindung dieser Modelle untereinander und mit dem Dialogmodell. Dadurch wird eine vollständigere Beschreibung der Interaktion ermöglicht, bei der die Anwendungslogik durch die Domänenobjekte verknüpft ist.

Gerade Aufgabenmodelle haben in der Mensch-Maschine Interaktion eine längere Tradition. So stellte Moran 1983 einen Ansatz für Mappings zwischen Aufgaben der externen Welt und internen Systemaufgaben vor: die *External Internal Task Analysis* (ETIT) [114]. Weitere Grammatiken wie die Task-Action Grammar (TAG) von Payne und Green [150] oder die User Action Notation (UAN) von Hartson et al. folgten [61]. Solche Grammatiken erlauben zwar die formale Analyse der Interaktion zwischen Benutzer und System, doch ihre Nutzbarkeit ist begrenzt. Wie van der Veer und van Welie ausführten, sind diese Methoden meist nur auf einen Ausschnitt der Interaktion bezogen und durch ihre textbasierte und komplexe Syntax schwierig zu erlernen und anzuwenden [200]. Grafische Notationen bieten eine intuitive Alternative.

3.5.1 ConcurTaskTrees

Grafische Methoden stellen eine intuitive und einfache Alternative zu komplexen textuellen Notationen dar. Als Erste stellten Annett und Duncan 1967 die hierarchische Task Analyse[4] vor, bei der Tasks durch eine Baumstruktur gegliedert werden [6]. Hierauf aufbauend entwickelte die Gruppe um Paternò die verbreitete Notation ConcurTaskTrees [147]. Sie definiert vier Arten von Tasks: Benutzertasks, die kognitiv oder perzeptiv sein können, Interaktionstasks, welche die Interaktion des Benutzers mit dem System repräsentieren, Anwendungstasks, in denen das System Aufgaben erledigen kann und abstrakte Tasks, mit denen zusammengesetzte Tasks beschrieben werden.

Mit ConcurTaskTrees werden hierarchische Strukturen aufgebaut. Die Knoten repräsentieren Tasks und stehen entweder in hierarchischen Beziehung oder in einer temporalen Beziehung zueinander. Eine temporale Beziehung wird auf der gleichen Ebene durch Verbindungen mit Operatoren annotiert. Die Operatoren entstammen LOTOS (*Language of Temporal Ordering Specifications*) [76, 67].

In Abbildung 3.7 wird der MP3-Player mit ConcurTaskTrees beschrieben. Der oberste Task ist abstrakt. Unter diesem befinden sich drei Tasks für das Abspielen von Musik, die Regelung der Lautstärke und die Navigation in der Playlist. Sie sind parallel aktiv, was durch die Verknüpfung mit `|||` angezeigt wird.

Im rechten Teilbaum unterbricht die Benutzeraktion `Pause` den abstrakten Task `Play`. Genauso unterbricht `Stop` Pause und Play. Eine Modellierung des Wiederaufnehmens aus dem Pausemodus ist jedoch komplexer, da in hierarchischen Bäumen eine Zyklusbildung durch Rückverbindungen verboten ist. Durch das rekursive Aufrufen von Play lässt sich diese Funktionalität nachbilden.

Die Tasks für die Regelung der Lautstärke und die Navigation in der Playlist sind fast identisch gestaltet. Ein Benutzer kann entweder lauter oder leiser regeln (`Next` bzw. `Previous` in der Playlist). Dies wird durch das Oder-Symbol `[]` angezeigt. Jeder dieser zwei Tasks besteht wiederum aus zwei Subtasks, einem Task für die Benutzeraktion und einem Systemtask. Sobald die Benutzeraktion ausgeführt wurde, setzt sie der folgende Systemtask in der Anwendung um. Die Tasks sind mit dem Operator `>>` verbunden, was bedeutet, dass nach erfolgreicher Beendigung des linken Tasks mit dem rechten Task begonnen wird.

Sieben Jahre nach der Arbeit zu ConcurTaskTrees entstand TERESA (Transformation Environment for inteRactivE Systems representAtion) durch die Zusammenarbeit von Berti et al. mit Paternò [17].

TERESA ist eine Entwicklungsumgebung mit einem grafischen Editor [115], die durch mehrere Transformationsschritte aus einer Taskbeschreibung in ConcurTaskTrees fertige Benutzerschnittstellen erzeugen kann. Begonnen wird mit einem allgemeinen und abstrakten Task-Modell in Form von ConcurTaskTrees, aus

[4] Späterer Repräsentant: Die Arbeit von Shepard von 1989, die *Hierarchical Task Analysis* (HTA) [176]vorstellt.

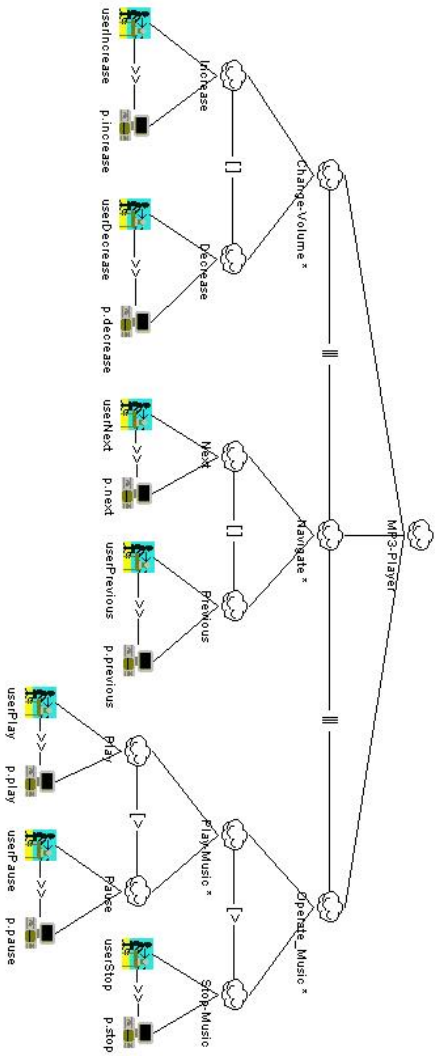


Abbildung 3.7: Die Aufgaben des MP3-Players mit ConcurTaskTrees.

dem die konkreten User Interfaces für verschiedene Plattformen abgeleitet werden können. Durch dieses Verfahren kann sich der Anwendungsdesigner zunächst auf die logischen Aspekte konzentrieren. Die plattformspezifischen Details werden erst in den späteren Arbeitsschritten betrachtet. Auf allen Abstraktionsebenen lässt sich die Benutzerschnittstelle XML-basiert speichern, wodurch ein Einstieg auf jeder Stufe auch durch fremde Werkzeuge erleichtert werden soll.

TERESA verlangt ein Top-down Vorgehen. Aus einer möglichst hohen Abstraktionsstufe werden durch die Transformationsschritte die konkreteren Benutzerschnittstellen erzeugt. Dabei werden verschiedene Plattformen unterstützt, denn TERESA erzeugt primär Oberflächen für das Web. Standardmäßig erzeugt es die allgemeine Spezifikation als XHTML-Code, im letzten Transformationsschritt kann für Pocket PC oder Mobiltelefone transformiert werden. Zusätzlich existiert eine Unterstützung von VoiceXML, um den Ansatz multimodaler zu gestalten.

Die Entwickler gingen davon aus, dass es weder möglich noch wünschenswert ist, alle Arten von Interaktionselementen für alle Plattformen anzubieten. So wird ein Mobiltelefon nicht Drag-and-Drop unterstützen und eine Webseite in der Regel keine Spracheingabe anbieten. Mittlerweile widerlegen das iPhone und barrierefreie Webseiten diese Prämisse. Bei der Transformation werden Tasks automatisch nach Heuristiken gruppiert. Auch bei den weiteren Schritten vom Dialog- zum Präsentationsmodell lassen sich Heuristiken für die Zielplattformen anwenden oder (z. B. durch einen Gestalter) die Transformationen anpassen, indem Gruppierung oder Stil (Farben, Schriftschnitte) geändert werden können.

ConcurTaskTrees sind eine mächtige und zugleich einfach anwendbare Notation zur Beschreibung von Aufgabenmodellen. Durch die zur Verfügung stehenden Transformationen in der Entwicklungsumgebung TERESA ist es mit mäßigem Aufwand möglich, Dialog- und Präsentationsmodelle abzuleiten und Anwendungen für verschiedene Zielplattformen zu erzeugen. Die Notation ist nicht einfach erweiterbar, bietet aber eine Ausdrucksstärke und Universalität, bei der Erweiterungen unnötig sein sollten. Sie ist meist verständlich, lediglich die Anwendung von Rekursion zur Wiederholung von Tasks ist für manche Entwickler ungebräuchlich. Doch die Werkzeugunterstützung durch TERESA ist ein großer Vorzug der ConcurTaskTrees. ConcurTaskTrees haben vor allem im Bereich der Task-Modellierung eine Verbreitung gefunden.

In Abbildung 3.8 zeigt die Einordnung der ConcurTaskTrees im Klassifikationsframework. Aus dem Aufgabenmodell wird zuerst das Dialog- und anschließend ein plattformspezifisches Präsentationsmodell abgeleitet. Ein Nachteil ist, dass die Gerätebeschreibungen nur im Werkzeug definiert sind. Auch das Fehlen eines Benutzer- und eines Umgebungsmodells bedeutet, dass multiple Präsentationsmodelle zum Ausgleich erzeugt werden müssten. Sie wären bis auf wenige Anpassungen identisch, was einen Mehraufwand bei Erstellung und Pflege der Benutzerschnittstellen bedeutet. Ein großer Nachteil ist vor allem, dass nur eine minimale Anbindung eines Domänenmodells existiert. Es gibt keine Automatismen, die Konsistenz zwischen den verwendeten Domänenobjekten und dem

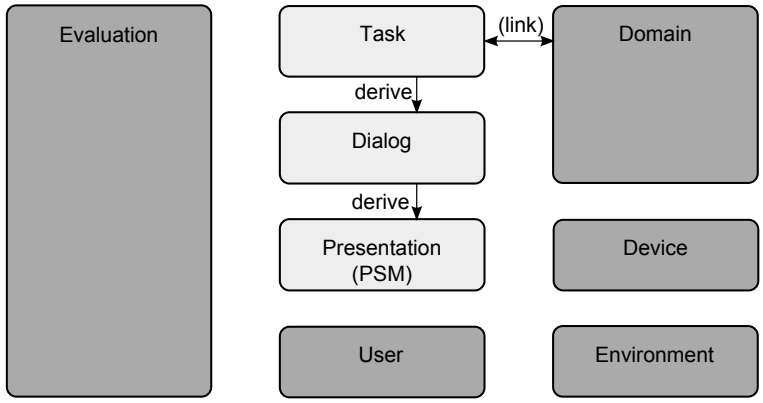


Abbildung 3.8: ConcurTaskTrees im Klassifikationsframework.

Aufgabenmodell herstellen. Das Fehlen eines Evaluationsmodells ist jedoch kein Nachteil und kann durch später besprochene Notationen ergänzend ausgeglichen werden.

3.5.2 USer Interface eXtensible Markup Language (UsiXML)

Die USer Interface eXtensible Markup Language (UsiXML) ist eine Notation, die von der Arbeitsgruppe um Vanderdonckt 2004 vorgestellt wurde und mehrere Entwicklungswerkzeuge umfasst [196]. UsiXML stützt sich stark auf die Erkenntnisse des Cameleon Projekts und legt auch das Cameleon Framework zu Grunde. Mittlerweile wird UsiXML von einem eigenen Konsortium standardisiert [194, 195].

Ein großes Anliegen dieses Konsortiums ist es, Werkzeuge für den Entwicklungs- und Transformationsprozess bereitzustellen. Sie sollen *multi-path development* unterstützen. Klassische Top-down Entwicklung von abstrakten zu konkreten Modellen soll ebenso möglich sein wie Bottom-up oder Middle-out Entwicklung oder das Arbeiten an allen Modellen (wide-spreading) zugleich. IdealXML ist ein Beispiel für ein Werkzeug. IdealXML ist ein grafischer Editor für das Aufgaben-, das Domänen- und das abstrakte UI-Modell [112]. Mit GrafiXML existiert ein weiterer grafischer Editor für die Erzeugung von Benutzerschnittstellen aus dem Aufgabenmodell [97]. Der Editor kann aus einem Aufgabenmodell automatisch verschiedene Implementierungen wie Java Swing oder HTML erzeugen.

Um diese breite Unterstützung bieten zu können, ermöglicht UsiXML auf verschiedenen Abstraktionsstufen zu modellieren und anschließend Transformatio-

nen in andere Stufen vorzunehmen. Es werden alle Modelle aus dem Klassifikationsframework, abzüglich des Dialog- und Evaluationsmodells, unterstützt. Zusätzlich werden Modelle für Mappings, Transformationen und Ressourcen eingeführt. Die Modelle werden der Reihe nach kurz erläutert.

Das *Taskmodell* entspricht dem Aufgabenmodell aus dem Klassifikationsframework. Es basiert auf ConcurTaskTrees, die um Benutzerstereotypen angereichert sind. So lassen sich Tasks für verschiedene Stereotypen gruppieren.

Im Domänenmodell lassen sich Klassen mit Attributen und Methoden beschreiben. Zusätzlich sind Relationen wie Aggregation und Generalisierung möglich.

Das Präsentationsmodell ist entsprechend dem Cameleon Framework in ein Abstract User Interface (AUI) und Concrete User Interface (CUI) unterteilt. Das AUI definiert schachtelbare Container zur Gruppierung von Elementen, sowie Eingabe- und Ausgabeelemente. Das CUI enthält spezifische Elemente für drei Modalitätskategorien. Für GUIs sind Elemente wie Fenster, Dialogboxen, Radiobuttons, oder Fortschrittsbalken vorhanden. Für Sprache stehen Ein- und Ausgabeelemente zur Verfügung. Darüber hinaus sind Elemente für 3D-Interaktion vorhanden, wie Buttons, Schieberegler, Bäume und Text.

Das vorhandene Kontextmodell beinhaltet Submodelle für die Umgebung, die Plattform und den Benutzer. Die Umgebung wird sehr einfach durch die drei Attribute `lightingLevel`, `isNoisy` und `isStressing` beschrieben. Für mögliche Plattformen existieren eine Vielzahl von Attributen, die jedoch primär auf GUIs ausgelegt sind. Benutzer werden über ihre Erfahrung und Motivation beschrieben.

Das Mappingmodell beschreibt durch Abstraktion, Konkretisierung, Übersetzung, Kontext und Update-Verbindungen die Beziehungen zwischen den verschiedenen Modellen.

UsiXML würde nicht dem Cameleon Framework entsprechen, wenn es keine Transformationsmöglichkeit von abstrakten Tasks zu konkreten Präsentationselementen bereitstellen würde. Durch ein Transformationsmodell lassen sich Regeln mit Bedingungen für diese Mappings erstellen. Entwicklungspfade für verschiedene Zielplattformen oder Benutzerstereotypen gruppieren dabei die Regeln.

Schließlich existiert noch ein Ressourcenmodell, das je nach Kontext verschiedene Übersetzungen erlaubt. Durch das Ressourcenmodell lassen sich z. B. unterschiedlich lange Texte je nach Displaygröße für verschiedene Geräte vorsehen. Das Modell ist ebenfalls weitgehend an GUIs orientiert und verfügt über Eigenschaften wie Tooltips oder Tastaturkürzel für einzelne Ressourcen.

UsiXML ist eine ausdruckstarke Notation, die universell einsetzbar ist. Sie läßt sich um neue Modalitäten erweitern, allerdings ist dazu eine Erweiterung der Notation selbst notwendig. Da sie zur Beschreibung von Interaktionsabläufen auf ConcurTaskTrees aufbaut, ist sie ebenso verständlich und damit eine angemessene Notation zur Modellierung. Eine wachsende Werkzeugunterstützung fördert die noch mäßige Verbreitung.

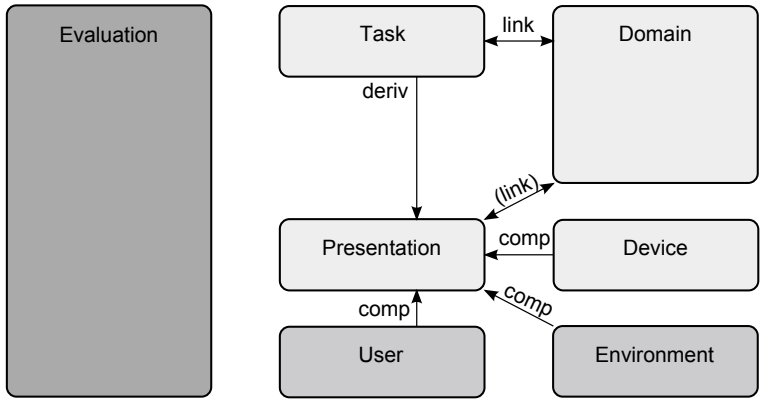


Abbildung 3.9: UsiXML im Klassifikationsframework.

Abbildung 3.9 zeigt UsiXML im Klassifikationsframework. UsiXML bietet den großen Vorteil, alle Modelle für Interaktion abzudecken. Durch das Transformationsmodell von UsiXML besteht schon innerhalb der Notation eine Möglichkeit, um Regelsätze für Transformationen aufstellen zu können. UsiXML unterstützt durch die Elemente des CUI-Modells insbesondere GUIs, nachrangig 3D- und Sprachschnittstellen. Diese können dank des Kontextmodells für verschiedene Einsatzzwecke, Benutzerstereotypen und Plattformen beschrieben werden. Von Nachteil ist dabei, dass Benutzer- und Umgebungsmodell sehr einfach aufgebaut sind, weshalb sie im Diagramm auch leicht eingefärbt sind.

3.5.3 Unified Modeling Language (UML)

Die Unified Modeling Language (UML) ist eine Sprache zum Modellieren der Strukturen von Systemen und ihrem Verhalten. Der Standard wird von der Object Management Group (OMG) gepflegt und ist mittlerweile in Version 2.2 verfügbar [141]. UML ist die am weitesten verbreitete Sprache zur Modellierung von Software und dient auch meist als Grundlage für Techniken der modellbasierten Softwareentwicklung.

Mitte der 1990er Jahre begannen Anstrengungen, um aus der Vielzahl von bestehenden Modellierungssprachen für objektorientierte Programmierung einen einheitlichen Standard zu schaffen [102]. Eine Reihe verschiedenster Beschreibungssprachen floss im Lauf der Zeit in UML ein. Es finden sich in UML z. B. Teile von Flussdiagrammen, die noch sehr ähnlich wie 1973 verwendet werden [123], Elemente aus Petrinetzen, die zur Beschreibung von Nebenläufigkeit dienen [153] oder Elemente der oben vorgestellten Zustandsdiagramme. Der große

Vorteil der UML ist, dass alle diese Konzepte durch eine Metasprache beschrieben werden können und damit untereinander verknüpfbar sind. Die *Meta Object Facility* stellt einfache, grundlegende Konzepte bereit, mit denen die Elemente der Sprachpakete beschrieben werden [137].

Die wichtigen Elemente für diese Arbeit werden in der *Superstructure Specification* festgelegt [142]. Dort werden die beiden Bereiche *Structure* und *Behavior* unterschieden. Structure enthält Sprachpakete mit denen statische Aspekte von Systemen beschrieben werden, davon werden in dieser Arbeit lediglich Klassendiagramme verwendet. Behaviors beschreiben das dynamische Verhalten eines Systems zur Laufzeit. In ihrem Bereich sind mehr Sprachpakete relevant, hiervon kommen Use-Case-Diagramme, Aktivitätsdiagramme, Zustandsdiagramme und Sequenzdiagramme zum Einsatz.

Durch UML lassen sich die relevanten Modelle für interaktive Benutzerschnittstellen abdecken. Das Domänenmodell ist mit Klassendiagrammen gleichzusetzen. Mit Klassendiagrammen können Klassen, ihre Attribute, Methoden mit ihren Parametern und Vererbungshierarchien beschrieben werden. Zusätzlich ermöglichen sie auch Beziehungen zwischen Klassen wie Aggregation oder Komposition. Auch Assoziationen wie in Entity-Relationship Modellen sind möglich um die Multiplizität einzelner Attribute beschreiben zu können.

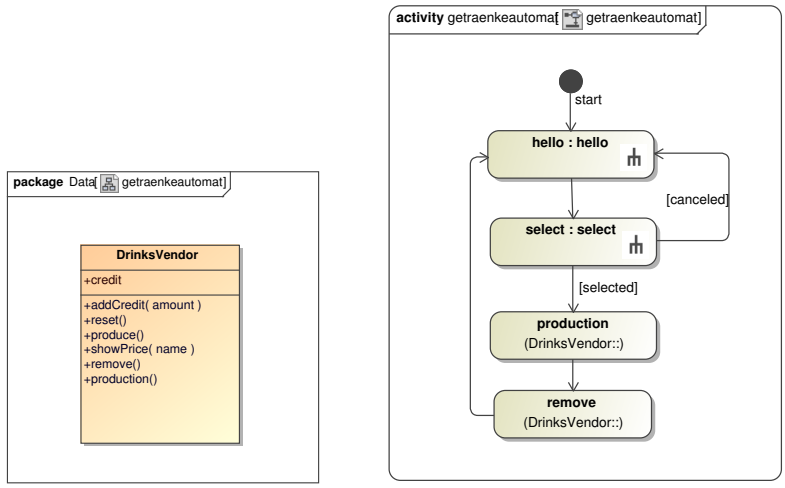
Aktivitätsdiagramme können als Aufgabenmodell eingesetzt werden. Genau genommen entspricht eine Aktion innerhalb der Diagramme gerade der Definition eines Tasks. Diese Diagramme beinhalten eine Ablaufsteuerung und können zusätzlich Nebenläufigkeit, das Abbrechen ganzer Bereiche und den Fluss von Objekten zwischen Aktionen beschreiben. Analog zu ConcurTaskTrees gibt es die Möglichkeit, hierarchische Diagramme aufzubauen, da eine Aktion als Beschreibung ihres Verhaltens wieder ein Aktivitätsdiagramm enthalten darf.

Im Gegensatz zu ConcurTaskTrees und UsiXML existiert mit den Zustandsdiagrammen ein eigener Diagrammtyp zur Beschreibung des Verhaltens von einzelnen Dialogen. Bei der Modellierung mit UML wird das Dialogmodell aus diesem Grund nicht aus dem Aufgabenmodell abgeleitet, sondern eigens erstellt. Zustandsdiagramme entsprechen weitgehend den in Abschnitt 3.4.3 beschriebenen Diagrammen. Sie können ebenfalls hierarchisch aufgebaut werden, um Parallelität abzubilden, und enthalten noch einige UML-spezifische Erweiterungen.

Eine eigene Diagrammart für ein Präsentationsmodell ist in UML nicht vorgesehen, auch Benutzer-, Umgebungs- oder Gerätemodelle fehlen. Sie lassen sich allerdings durch das Einbinden bestehender Modelle oder durch Beschreibung solcher Modelle in UML bereitstellen. Durch das Beschreiben als Sprachpakete für UML bleibt dabei die Kompatibilität innerhalb der Sprache erhalten.

UML bietet jedoch einige zusätzliche Diagramme, die weitere Sichten auf das Modell liefern. Für diese Arbeit sind Use-Case-Diagramme und Sequenzdiagramme relevant.

Use-Case-Diagramme zeigen, was ein System für Benutzer leistet. Sie werden meist eingesetzt um das Verhalten von Benutzerschnittstellen zu beschreiben.



(a) Klassendiagramm (b) Aktivitätsdiagramm

Abbildung 3.10: Links das Klassendiagramm des Getränkeautomaten, rechts das Aktivitätsdiagramm.

Jeder Use-Case (Anwendungsfall) kapselt eine Menge von Aktionen in sich. Ein Use-Case ist z. B. beim Getränkeautomat das Kaufen eines Getränks. Damit steht eine Möglichkeit zur Verfügung, komplexe Aufgaben in einem Use-Case zusammenzufassen und Aufgaben miteinander zu verbinden.

Sequenzdiagramme zeigen, wie verschiedene Elemente zur Laufzeit miteinander kommunizieren. Sie beschreiben die Arten von Nachrichten, die gesendet werden können, wann dabei neue Objekte im Domänenmodell erstellt werden oder wann ein Objekt zerstört wird. Damit eignen sie sich für die Beschreibung der Abfolge von Interaktionen zwischen dem Benutzer, dem System als Ganzem und seinen internen Objekten. Zusätzlich werden Sequenzdiagramme oft zur Spezifikation von Testfällen genutzt [85, 88].

Das Fallbeispiel des Getränkeautomaten soll für UML als Beispiel dienen. Das Domänenmodell wird in Abbildung 3.10a als einfache Klasse abgebildet. Sie besitzt unter anderem Methoden, um das Guthaben des Benutzers beim Automat zu erhöhen und das Geld zurückzufordern. Als Aufgabenmodell dient ein Aktivitätsdiagramm, welches in Abbildung 3.10b zu sehen ist.

Die beiden Zustandsdiagramme in Abbildung 3.11 und 3.12 beschreiben das Verhalten der Aktionen `hello` und `select` aus dem Aktivitätsdiagramm. Im Aktivi-

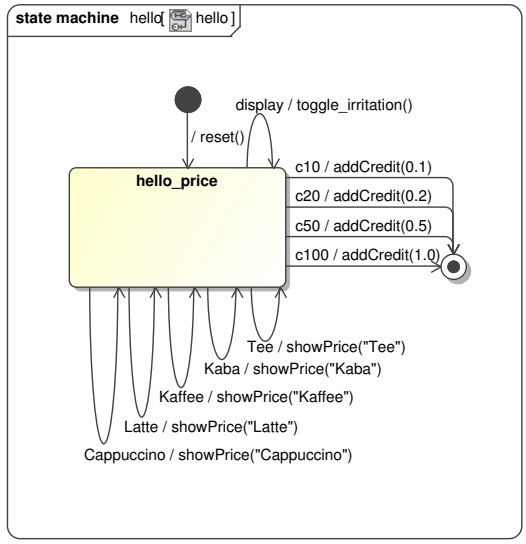


Abbildung 3.11: Zustandsdiagramm zur Beschreibung des Verhaltens der Aktion *hello*. Es zeigt die Begrüßung mit Geldeinwurf.

tätsdiagramm wird durch das Gabelungssymbol angezeigt, dass ein erweiterndes Diagramm vorhanden ist. In beiden Zustandsdiagrammen gibt es eine Reihe von Transistionen, die Benutzerinteraktionen als auslösende Ereignisse besitzen. Die annotierten Befehle sind mit dem Domänenmodell verbunden.

Fazit

UML ist eine sehr ausdrucksstarke und universell einsetzbare Notation. Durch ihr Metamodell lässt sie sich um neue Sprachpakete (Diagramme) erweitern. Die grafische Notation ist verständlich und es gibt Notationsvarianten für viele Sonderfälle, so dass sich in der Regel eine angemessene Umsetzung in ein Modell finden lässt.

In Abbildung 3.13 ist UML im Klassifikationsframework eingeordnet. Die Stärken liegen deutlich bei den Modellen im oberen Bereich. UML bietet mit Aktivitäts- und Zustandsdiagrammen deutlich mächtigere Aufgaben- und Dialogmodelle als UsiXML. UML bietet mehr Varianten zum Modellieren der Aufgabenmodelle als ConcurTaskTrees [130]. Ebenso wie bei UsiXML bleibt man innerhalb

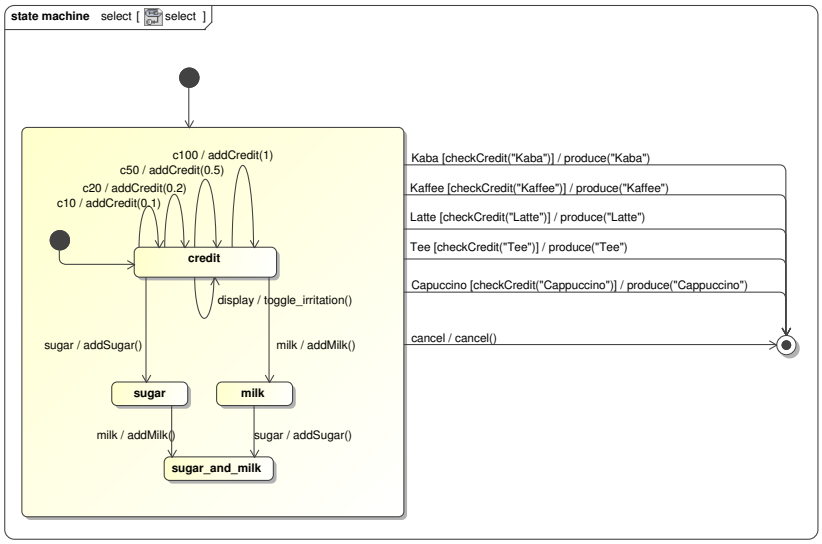


Abbildung 3.12: Zustandsdiagramm zur Beschreibung des Verhaltens der Aktion *select*. Es zeigt die Auswahl eines Getränks.

einer Sprachfamilie, da die verschiedenen Diagramme alle durch die selbe Metasprache beschrieben werden. Bei UML werden die Elemente anhand von IDs miteinander verknüpft, so dass sie bei einer Veränderung automatisch aktualisiert werden.

Für UML existiert eine große Auswahl an Modellierungswerkzeugen, von rein grafischen Editoren hin zu Systemen, die Codegenerierung aus Diagrammen oder Roundtrip-Engineering bieten. Dank dieser weitreichenden Werkzeugunterstützung stellt UML einige weitere Vorzüge bereit. So lassen sich Prüfungen durchführen, ob Elemente verknüpft sind, so dass z. B. eine verknüpfte Methode nicht einfach gelöscht werden kann. UML ist eine sehr weit verbreitete Beschreibungssprache, die in vielfältigen Bereichen Anwender besitzt.

Das Fehlen von Präsentations- und Kontextmodellen ist nur bedingt ein Nachteil für den Einsatz von UML. In den bereits vorgestellten Notationen UsiXML und ConcurTaskTrees wird das Präsentationsmodell aus dem Aufgabenmodell abgeleitet. Bei UML ließe es sich ebenfalls aus Aufgaben- und Dialogmodell ableiten. Die anderen Modelle lassen sich durch Einbinden bestehender Ansätze oder Abdecken in UML ergänzen.

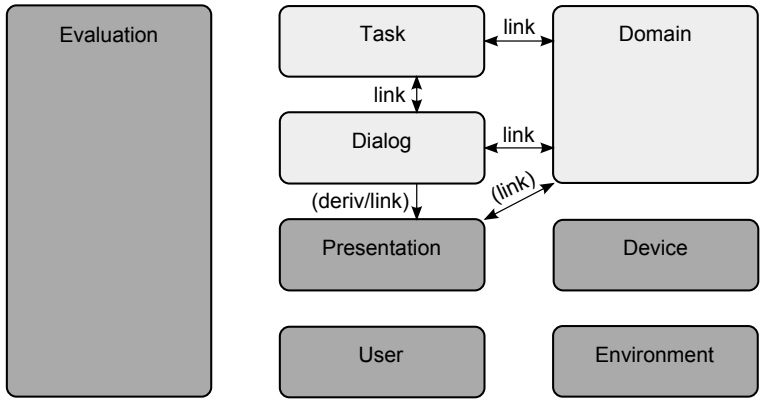


Abbildung 3.13: UML im Klassifikationsframework.

3.6 Modelle für Kontext, Evaluation und einzelne Modalitäten

Nachdem im vorigen Abschnitt abstrakte Modelle für Interaktion vorgestellt wurden, sollen in diesem Abschnitt die übrigen spezialisierten Modelle für Kontext, Evaluation und einzelne Modalitäten kurz eingeführt werden. Für die einzelnen Bereiche werden Überblicke und Repräsentanten angegeben. Durch die Ergänzung mit diesen Modellen lassen sich alle Aspekte der modellbasierten Entwicklung interaktiver Anwendungen abdecken.

Der Kontext einer Anwendung setzt sich aus Benutzer-, Geräte- und Umgebungsmodell zusammen.

3.6.1 Beschreibung von Benutzereigenschaften

Ein Benutzermodell beschreibt die verschiedenen Eigenschaften eines Benutzers. Dazu gehören unveränderliche Attribute, wie Größe oder Geschlecht. Für viele Fälle sind gerade Einschränkungen wie Sehschwächen, Farbblindheit oder reduzierte Beweglichkeit wichtig. Benutzerschnittstellen sollten sich anpassen um dem Benutzer den Zugang zu Informationen und die Interaktion möglichst einfach zu machen. Dynamisch veränderliche Attribute sind z. B. die Stimmung oder die Aufmerksamkeit des Benutzers. Abhängig von der Aufmerksamkeit können z. B. Spracheingaben nicht mit voller Sicherheit erkannt werden oder der Benutzer könnte auch abgelenkt zu einem anderen Thema antworten.

Ein frühes Benutzermodell wurde 1983 von Card, Moran und Newell in ihrem Buch „The Psychology of Human Computer Interaction“ vorgestellt [28, 31]. Es besitzt vier Ebenen: die physikalische, die kognitive, die konzeptuelle und die Task-Ebene. Die physikalische Ebene beinhaltet dabei die Interaktion des Benutzers mit Eingabe- und Ausgabegeräten. Die kognitive Ebene beschreibt die Kapazitäten des Benutzers, die zur Informationsverarbeitung vorhanden sind. Auf der konzeptuellen Ebene steht das Modell, welches ein Benutzer durch die eigene Wahrnehmung für sich konstruiert. Auf der Task-Ebene werden schließlich Aufgaben und Ziele beschrieben.

Ein Beispiel soll die verschiedenen Ebenen illustrieren. Ein Benutzer möchte einen Kaffee trinken (Ziel). Um sein Ziel zu erreichen will er einen Kaffee von einem Automaten kaufen. Die Aufgabe besteht in der Auswahl der richtigen Sorte Getränk, dem Bezahlen und dem Entnehmen des fertigen Produkts. Auf der konzeptuellen Ebene besitzt der Benutzer eine Vorstellung von der Funktion des Kaffeeautomaten: Gegen die Entrichtung des Preises wird dieser ihm ein Getränk herstellen. Auf der kognitiven Ebene kommen Gesetzmäßigkeiten über die Kapazität des Kurzzeitgedächtnisses ins Spiel [110]. Wenn zuviele Getränke zur Auswahl stehen, wird es schwerer das richtige Getränk zu wählen und die restlichen Arbeitschritte im Gedächtnis zu behalten. Auf der physikalischen Ebene erfolgt die Bedienung des Geräts durch das Drücken der richtigen Tasten und den Geldeinwurf.

3.6.2 Gerätemodell

Durch das Gerätemodell lassen sich Profile für verschiedene Hardwareplattformen, sogenannte Geräteprofile, definieren. In einem Profil werden die Eigenschaften für die Ein- und Ausgabefähigkeiten der Plattform gesammelt. Soll ein Präsentationsmodell in ein plattformspezifisches Modell transformiert werden, so werden die entsprechenden Fähigkeiten des Endgeräts berücksichtigt. Beispielsweise müssen Geräte mit kleinen Displays andere Steuerelemente aufweisen als Geräte, die viel Platz für eine Anwendung bieten können.

Ein Ansatz für solche Beschreibungen sind die *Composite Capability/Preference Profiles* (CC/PP), die durch eine Ontologie die Eigenschaften von Geräten formal beschreiben [214]. Die Ontologie wird dabei mit Hilfe des *Resource Description Framework* (RDF) in XML erstellt [212]. Einzelne Geräte werden durch eine Ansammlung ihrer Attribute gekennzeichnet. Ein Ausgabegerät besitzt z. B. Attribute für die Auflösung in Pixeln.

3.6.3 Umgebungsmodell

Umgebungsmodelle beschreiben die Umgebung, in der die Interaktion zwischen Benutzer und System stattfindet. Es existieren unterschiedlich komplexe Model-

le, da der Einfluss der Umgebung auf Anwendungen von der Anwendung selbst und allgemein gültigen Faktoren bestimmt wird.

Im Zusammenhang mit UsiXML wurde in Abschnitt 3.5.2 bereits ein einfaches Umgebungsmodell vorgestellt (vgl. [196]). Es ist in das Kontextmodell von UsiXML integriert und beschreibt die Umgebung durch drei Attribute, nämlich durch den `lightingLevel`, der durch eine Zahl dargestellt wird, sowie `isNoisy` und `isStressing`. Die letzten beiden Attribute sind Wahrheitswerte. Dieses einfache Modell genügt bereits, um eine Reihe von unterschiedlichen Ausprägungen von Benutzerschnittstellen beschreiben zu können.

Calvary et al. haben weitere Attribute von Umgebungsmodellen beschrieben [24]. Ein Modell kann Informationen zu vorhandenen Objekten, Personen und Vorgängen enthalten, die peripher zur aktuellen Aufgabe eines Benutzers gehören, sich aber auf sein Verhalten auswirken können. Nach den Autoren sollen die Attribute durch Domänenanalysten festgelegt werden, die entscheiden, welche Informationen für eine Anwendung relevant sind.

3.6.4 Modelle zur Evaluation

Evaluationsmodelle beschreiben, wie eine Reihe von Eigenschaften der Benutzerschnittstelle gemessen werden können. Es ist möglich, die Geschwindigkeit des Benutzers bei der Erfüllung einer bestimmten Aufgabe festzustellen, zu messen, wie angenehm Benutzer eine Benutzerschnittstelle finden, wie viele Fehler Benutzern im Mittel bei einer Aufgabe unterlaufen und vieles mehr.

Es existieren mehrere Evaluationsmodelle, von denen GOMS als wichtiger Repräsentant vorgestellt wird. GOMS wurde 1983 von Card, Moran und Newell entwickelt und steht für *Goals*, *Operators*, *Methods* und *Selection* [28, 45, 177]. Im Einzelnen bedeuten diese Elemente folgendes:

- *Goals* sind die Ziele des Benutzers, die er bei einer Aufgabe erreichen möchte. Sie werden in GOMS auch als eine Art Knoten dargestellt, von denen aus ein Benutzer bewerten kann, was getan werden soll und zu denen bei Fehlern zurückgekehrt werden kann.
- *Operators* sind die atomaren Aktionen, die Benutzer ausführen, wenn sie das System benutzen. Sie beeinflussen das System (z. B. durch Ausführen einer Aktion) oder den Benutzer (z. B. durch Information, die angezeigt wird).
- *Methods* kennzeichnen Wege zum Ziel. In GOMS werden Methods durch das kleingeschriebene Schlüsselwort *goal* gekennzeichnet.
- *Selection* kennzeichnet die Wahlmöglichkeit zwischen den verschiedenen *Methods*.

GOMS kommt aus dem Bereich der kognitiven Modellierung. Mit GOMS lassen sich gut hierarchische Planungen darstellen und Vorgänge in atomare Abfolgen zerlegen.

```
1  Goal: Kaffee mit Milch und Zucker kaufen
2    [select goal: geld-einwerfen     # method 1
3                  zucker-waehlen     # method 1
4                  milch-waehlen      # method 1
5                  kaffee-druecken    # method 1
6            goal: geld-einwerfen     # method 2
7                  milch-waehlen      # method 2
8                  zucker-waehlen     # method 2
9                  kaffee-druecken    # method 2
10   ]
```

Listing 3.3: Kaufen eines Kaffees an einem Getränkeautomat mit GOMS.

Das Beispiel in Listing 3.3 zeigt eine Modellierung des Getränkeautomaten mit GOMS. In diesem Beispiel gibt es nur ein Ziel: der Benutzer möchte einen Kaffee mit Milch und Zucker kaufen. Dazu gibt es zwei Wege, nach dem Geld einwerfen kann zuerst Zucker und dann Milch ausgewählt werden oder die Reihenfolge kann umgekehrt sein. Daraufhin muss in jedem Fall die Taste für den Kaffee am Automat gedrückt werden. Das Ziel hat zwei Teilziele, die durch `select` gewählt werden können (Selection). Die beiden Teilziele sind die Methods, die wiederum aus einzelnen Operators bestehen. Die Methods sind durch Kommentare (`# method 1`) gekennzeichnet.

GOMS bietet eine einfache Möglichkeit, Vorhersagen über die Zeit zu machen, die ein Benutzer für Aktionsabfolgen benötigt. Dazu werden die einzelnen Schritte erhoben und nach der Zeit gemittelt. Anschließend lassen sich bei Änderungen der Abläufe durch einfaches Aufsummieren der Zeiten Vorhersagen treffen.

3.6.5 Notationen für einzelne Modalitäten

Notationen für einzelne Modalitäten sind durch ihre Spezialisierung entsprechend mächtig auf ihrem Gebiet. Es gibt eine große Anzahl an solchen Beschreibungssprachen, deren Aufzählung nicht zielführend ist, deshalb werden nur wenige vorgestellt. Diese werden aus den drei großen Modalitätskategorien GUIs, Sprache und TUIs herausgegriffen.

Insgesamt haben Notationen für einzelne Modalitäten eine längere Geschichte. Angefangen bei Toolkits, die zur Vereinfachung der Programmierung von GUIs entstanden sind, entwickelten sich eigenständige Notationen. Ein frühes Beispiel ist UIL, die Motif User Interface Language, eine Sprache die GUIs für Motif[5] be-

[5]Motif entstand um 1989 und wird weiterhin gepflegt. Die letzte Version 2.3.2 stammt vom 12.6.2009.

schreibt. Sie war weit verbreitet und wurde auch von gängigen Werkzeugen unterstützt [51]. Tk ist dagegen ein um 1989 entstandenes plattformübergreifendes Toolkit, das auch über Anbindungen zu einer ganzen Reihe von Programmiersprachen verfügt. Vor allem durch seine frühe Verfügbarkeit und die Anbindung an Skriptsprachen fand Tk schnelle Verbreitung [197].

Seit den 1990ern wurden verschiedene neue Notationen für GUIs entwickelt. Zur abstrakten Beschreibung von GUIs wurden z. B. die User Action Annotation (UAN) von Hartson vorgestellt, die Interaktionen durch Maus und Tastatur als Eingabeoperationen beschreibt [61].

Constantine et al. stellten 2002 die Canonical Abstract Prototypes (CAP) vor, eine Notation, die GUIs auf dem Präsentations- und Dialogmodell beschreibt [36, 37]. Einzelne Teildialoge konnten hier noch nicht miteinander verbunden werden. Allerdings wird bereits eine grafische Notation benutzt.

Eine aktuelle Notation ist XUL (*XML User Interface Language*), die von Goodger et al. entwickelt wurde [57, 157]. XUL setzt auf der Gecko Engine von Mozilla auf und ermöglicht es, mit jedem Mozilla-basierten Webbrowser eine native Oberfläche zu erstellen.

Natürlich existieren neben Notationen für GUIs auch Notationen für andere Modalitäten. Im Sprachbereich ist VoiceXML weit verbreitet, eine Markup-Sprache des W3C für Sprachdialoge, die schon in der zweiten Version vorliegt [103]. VoiceXML findet zusätzlich zu reinen Sprach- bzw. Telefonieanwendungen auch Einsatz in Kombination mit Webseiten durch den Standard XHTML+Voice [10].

Für Tangible User Interfaces entstehen erst Notationen. Leland et al. stellten 2004 in einem Konzeptpapier die Grundlagen für TUIML vor, eine Beschreibungssprache für TUIs. TUIML ähnelt den vorgestellten abstrakten Modellen, denn sie verfügt ebenfalls über ein Aufgaben-, Domänen- und Repräsentationsmodell. Zusätzlich soll TUIML noch Elemente für die zu manipulierenden Objekte und ihre Eigenschaften enthalten. Diese Notation ist jedoch erst in der konzeptionellen Phase.

3.7 Bewertung

Die in Abschnitt 3.4 vorgestellten Grammatiken, Ereignissprachen, Produktionsregeln und Zustandsdiagramme sind zwar ausdrucksstarke Notationen, durch die mangelnde Anbindung der Domäne und weiterer Modelle genügen sie nicht den Anforderungen an eine universelle abstrakte Notation für Interaktionsanwendungen. Auch die im selben Abschnitt am Rande gestreiften neueren Notationen lösen diese Probleme nicht.

Es genügt nicht, durch Notationen wie Grammatiken, Ereignissprachen und Produktionsregeln eine formale Analyse zu ermöglichen, während gleichzeitig ihre Nutzbarkeit begrenzt bleibt. Diese Begrenzung soll durch grafische Notationen

überwunden werden [200]. Zentrale Modelle für Interaktionsanwendungen sind das Domänen-, Aufgaben- und Dialogmodell.

Die vorgestellten abstrakten Modelle setzten alle grafische Notationen ein. ConcurTaskTrees haben ihre Stärke bei der Beschreibung hierarchischer Tasks. Nachteilig sind die fehlende Anbindung eines Domänenmodells und die fehlenden Kontextmodelle für Geräte, Umgebung und Benutzer. Die hierarchische Modellierung ist manchmal ein Hindernis bei der Modellierung, da sie dazu zwingt für teils einfache Anwendungen tiefe Hierarchien erstellen zu müssen. Auch die Verwendung von Rekursion zur Wiederholung eines Tasks ist nicht intuitiv.

UsiXML deckt am meisten Modelle unter den vorgestellten Notationen ab. Die Möglichkeit, Mappings und Transformationen frei definieren zu können, ist ein großer Vorteil des Ansatzes. Ein gravierender Nachteil ist, dass einige Modelle nur rudimentär implementiert sind. Ein weiterer Nachteil ist die Vermischung verschiedener Modalitätskategorien beim Präsentationsmodell. An dieser Stelle ist eine Trennung der Modalitäten in verschiedene plattformspezifische Präsentationsmodelle wie z. B. bei ConcurTaskTrees vorteilhafter. Da UsiXML nicht erweiterbar ist, lassen sich später auch keine Elemente von neuen Geräten oder neuen Interaktionsverfahren ergänzen.

Die Unified Modeling Language (UML) deckt nur einen Teil aller Modelle ab. Allerdings ist ihre Ausdrucksstärke bei den Modellen deutlich größer als bei den anderen Notationen. Dies ist nicht verwunderlich, da aus vielen früheren Beschreibungssprachen Ideen in die UML übernommen wurden. Die UML ist die ausdruckstärkste Notation beim Domänenmodell. Im Vergleich zu ConcurTaskTrees ist sie in Bezug auf das Aufgabenmodell mindestens genauso mächtig, da sie über mehr Varianten zum Ausdrücken von Verhalten verfügt. Während andere Notationen ein Dialogmodell aus einem Aufgabenmodell ableiten, ist die UML die einzige Notation, die ein komplett eigenes Dialogmodell bietet. Eine weitere Stärke liegt in ihrer Verbreitung und der verfügbaren Werkzeugunterstützung. Das Fehlen von Präsentations- und Kontextmodellen ist nicht zwingend ein Nachteil, da diese aus anderen oben vorgestellten Notationen ergänzt werden können. Ein zusätzlicher Vorteil der UML ist ihre Skalierbarkeit und die Fähigkeit komplexe Anwendungen zu beschreiben. Auch Hierarchiebildung ist möglich. Dazu bietet UML eine Reihe von Alternativen wie Diagramme miteinander verbunden oder ineinander eingebettet werden können.

Tabelle 3.1 fasst die bewerteten Eigenschaften der vorgestellten Notationen nach den Kriterien aus Abschnitt 3.2.1 noch einmal zusammen. Die Verständlichkeit von Produktionssystemen hängt stark von der Gestaltung der Regeln ab und ist darum geklammert.

UML bietet die meisten Vorzüge und wird daher für diese Arbeit zur Beschreibung des Verhaltens und der Domänenobjekte von Interaktionsanwendungen ausgewählt. Die Entscheidung für UML begründet sich durch die Mächtigkeit der Notation zur Beschreibung von Aufgaben-, Dialog- und Domänenmodell. Ein nicht zu vernachlässigender Faktor ist auch die starke Verbreitung und Werk-

Kriterium	Grammatik	Produktionssysteme	Zustandsdiagramme	ConcurTaskTrees	UsiXML	UML
Erweiterbarkeit um Modalitäten	+	+	o	o	+	+
Angemessenheit	o	+	+	+	++	++
Verständlichkeit	o	(o)	++	+	+	++
Ausdrucksstärke	+	+	++	++	++	++
Universalität	+	+	++	++	++	++
Werkzeugunterstützung	-	-	+	++	+	++
Verbreitung	-	-	++	+	+	++

Tabelle 3.1: Zusammenfassung der Eigenschaften der vorgestellten Notationen. Bewertungsskala: --, -, o, +, ++

zeugunterstützung von UML. Die Vertrautheit durch den Einsatz im normalen Entwicklungsalltag reduziert die Hürden zum Einsatz der Notation zur Interaktionsmodellierung.

Das folgende Kapitel geht detaillierter auf die Modellierung mit UML ein. Im Anschluss wird eine Architektur zur Verwendung der Modelle beschrieben, bevor ein Kapitel die prototypische Realisierung beschreibt.

4 Modellieren von Interaktionsanwendungen mit UML

> Die Idee ist das Absolute, und alles Wirkliche ist nur Realisierung der Idee.
>
> *Georg Wilhelm Friedrich Hegel*
> *(1770 – 1831)*

Im vorangegangenen Kapitel wurde die Unified Modeling Language (UML) als Notationen für Interaktionsanwendungen ausgewählt. Die UML ist in ihren einzelnen Sprachpaketen deutlich mächtiger als ihre Konkurrenten. Mittlerweile ist sie außerdem eine so weit verbreitete Notation, dass sie als *lingua franca*[1] der Software-Ingenieure gilt. Es existieren mehrere Versionen, im April 2005 wurde von der Object Management Group der neue Standard UML 2.0 offiziell verabschiedet, dieser Arbeit liegt die Version 2.2 von 2009 zu Grunde [141, 142].

In diesem Kapitel wird detailliert beschrieben, wie man Interaktionsanwendungen mit UML modelliert. Ein Beitrag dieser Arbeit ist eine Verfahrensbeschreibung, die eine Anleitung zur Modellierung von Interaktionsanwendungen mit UML gibt. Es ist nicht Ziel dieses Kapitels, eine umfassende Einführung in UML zu geben, hierzu sei auf die Bücher von Jeckle et al. oder Kecher verwiesen [85, 88].

Da die UML aus einer Reihe von Notationen entstand, enthält sie mittlerweile 14 verschiedene Diagrammarten. Für diese Arbeit sind davon nur eine Auswahl relevant: Klassen-, Aktivitäts-, Use-Case und Sequenzdiagramme. Die folgenden Abschnitte sind nach diesen Diagrammarten gegliedert. Sie zeigen, welche Diagramme für ein Modell zum Einsatz kommen, und stellen die wichtigsten Elemente vor. Zur Illustration werden die beiden Fallbeispiele aus dem Grundlagenkapitel verwendet.

[1] Verkehrssprache, weit verbreitete Sprache

Abbildung 4.1 zeigt die Zusammenhänge zwischen den verschiedenen Diagrammen. Use-Case-Diagramme werden eingesetzt, um Aktivitätsdiagramme zu strukturieren. Aktivitätsdiagramme beschreiben den Ablauf von Tasks und verwenden Zustandsdiagramme, um einzelne Aktionen detailliert zu beschreiben. Ein Zustandsdiagramm kann zur Beschreibung eines Verhaltens wiederum ein Aktivitätsdiagramm verwenden. Beide, Aktivitäts- und Zustandsdiagramme, setzen Klassendiagramme ein, um auf Domänenobjekte zuzugreifen.

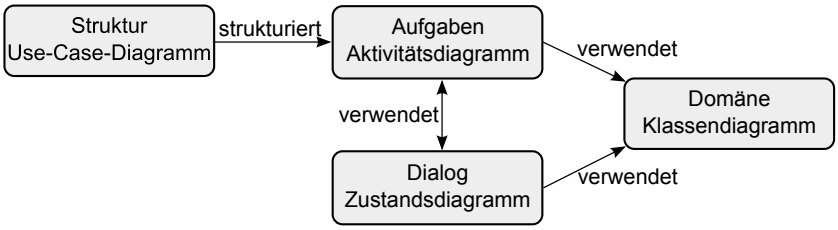


Abbildung 4.1: Zusammenhang zwischen den eingesetzten UML-Diagrammen.

Zum Ende des Kapitels wird auf die Modelle eingegangen, die von UML nicht direkt unterstützt werden. Es werden Ansätze gezeigt, mit denen insbesondere das Präsentationsmodell realisiert werden kann. Dazu wird auf den Kontext einer Anwendung eingegangen, also auf die Benutzer-, Umgebungs- und Gerätemodelle.

4.1 Domänenmodell

Durch das Domänenmodell sollen die Objekte, die in einem System vorhanden sind, definiert werden. Es legt fest, welche Aktionen mit Objekten ausgeführt werden können, welche Eigenschaften Objekte besitzen und welche Relationen zwischen ihnen bestehen. Andere Modelle greifen auf die Domänenobjekte zurück, um Aktionen in einer Anwendung zu bewirken. Dies können das Aufgaben-, Dialog- oder Präsentationsmodell sein.

Um die Domäne einer Anwendung zu beschreiben, werden Klassendiagramme eingesetzt. Jede Klasse beschreibt dabei den Typ einer Reihe von Objekten. Zur Laufzeit einer Anwendung werden Objekte der Klassen instanziiert, die anschließend von anderen Modellen verwendet werden können.

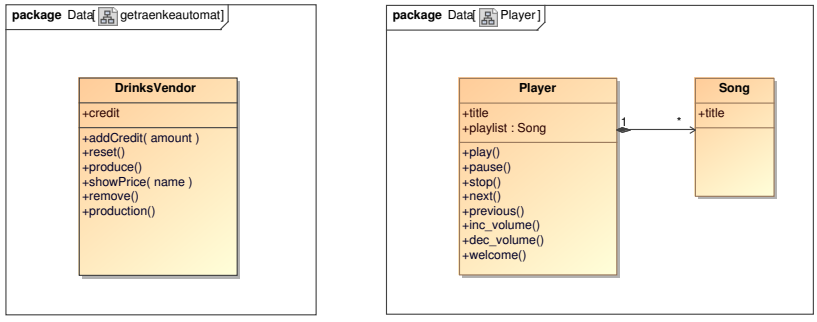
Klassendiagramme müssen sich dabei nicht auf die Klassen beschränken, die für andere Modelle wichtig sind, sie können vielmehr die komplette Anwendung und ihre internen Details beschreiben.

4.1.1 Klassendiagramm

Klassendiagramme dienen zur Beschreibung statischer Strukturen. Sie visualisieren dazu die wesentlichen Eigenschaften der verschiedenen Strukturen und die Beziehungen zwischen den Strukturen.

Ein Klassendiagramm enthält eine Reihe von Elementen: Klassen, Attribute und Operationen werden hier vorgestellt. Auch wenn es noch mehr Elemente gibt, genügen Klassen mit ihren Attributen und Operationen, um genug Funktionalität für das Domänenmodell zur Verfügung zu stellen. Assoziationen und Generalisierungen dienen zur Darstellung von Abhängigkeiten unter den Klassen und zur Beschreibung von Vererbungshierarchien. Sie besitzen keine Auswirkungen auf Benutzerinteraktionen, darum wird im Folgenden nur ein Teil ihrer Einsatzmöglichkeiten vorgestellt.

Klassen verkörpern Typen, denen Objekte zugeordnet werden. Ihre Attribute beschreiben die statischen Eigenschaften, die Objekte besitzen. Jede Klasse besitzt eine Liste mit Operationen, die auf ein Objekt der Klasse angewandt werden können. Jede Operation kann Parameter besitzen. In Abbildung 4.2a und 4.2b sind die Klassen des Getränkeautomats und des MP3-Players zu sehen. Der Getränkeautomat besitzt nur ein einzelnes Attribut, nämlich das Guthaben, das ein Benutzer einzahlen kann. Der MP3-Player hat zwei Attribute, den Titel des aktuellen Lieds und die Playlist. Beide Anwendungen besitzen eine Reihe von Operationen. Der Getränkeautomat hat z. B. eine Methode um Geld einzuzahlen, die auch einen Parameter übergeben bekommt: `addCredit(amount)`. Der MP3-Player hat dagegen nur Methoden ohne Parameter.



(a) Klassendiagramm Getränkeautomat (b) Klassendiagramm MP3-Player

Abbildung 4.2: Links das Klassendiagramm des Getränkeautomaten, rechts das des MP3-Players.

Das Klassendiagramm des MP3-Players zeigt, dass die Klasse `Player` mit der Klasse `Song` assoziiert ist. Dabei handelt es sich um eine Komposition. Ein Objekt der Klasse `Player` kann beliebig viele Objekte der Klasse `Song` besitzen. Die `Song`-Objekte bilden in diesem Fall die Playlist.

Nicht abgebildet sind Generalisierungsbeziehungen. Diese bezeichnen Vererbungshierarchien und zeigen, welche Klasse eine Ableitung einer allgemeineren Oberklasse ist.

4.1.2 Einsatz

Mit Klassendiagrammen lässt sich die gesamte Anwendungslogik eines Systems beschreiben. Wichtig für den Einsatz zur Modellierung von Interaktionsanwendungen ist, dass nicht einfach alle Klassen für die Interaktion verwendet werden. Es ist besser, die Komplexität einer Anwendung durch wenige Klassen, die als Fassade fungieren, zu verbergen. Zu diesem Zweck wird das *Facade*-Pattern eingesetzt [55].

Klassen stellen durch ihre Attribute Zustandsvariablen für die Anwendung zur Verfügung, die zur Laufzeit durch die Interaktion mit dem Benutzer verändert werden. In den Beispielen sind diese das Guthaben eines Benutzers und das aktuelle Lied.

Durch die bereitgestellten Operationen der Klassen kann von anderen Diagrammen aus auf die Anwendung zugegriffen werden. Damit lassen sich durch Benutzerinteraktionen Vorgänge im System starten oder beeinflussen.

4.2 Aufgabenmodell

Das Aufgabenmodell beschreibt die einzelnen Aufgaben, die ein Benutzer mit einer Anwendung erledigen kann. Es stellt eine abstrakte Sicht auf die gesamte Interaktion dar. Das Aufgabenmodell ist direkt und indirekt durch das Dialogmodell mit dem Präsentationsmodell verbunden.

In UML gibt es zwei Diagrammarten, die mit dem Aufgabenmodell zusammenhängen. Das Use-Case-Diagramm eignet sich für abstrakte Beschreibungen von Systemen und zur funktionalen Gliederung von Systemen. Darüber hinaus zeigen sie die Intentionen der Benutzer und die Erwartungen an das System auf, wie von Constantine beschrieben [35, S. 247–252]. Aktivitätsdiagramme beschreiben Abläufe und das Verhalten eines Systems. In ihnen können Aufgaben parallel oder sequentiell erledigt werden. Es gibt auch die Möglichkeit, dass ein Benutzer sich zwischen manchen entscheiden kann.

Zuerst werden kurz Use-Case-Diagramme vorgestellt, die zur Gliederung komplexer Anwendungen Verwendung finden. Im Anschluß folgen Aktivitätsdiagramme, die das allgemeine Verhalten eines Systems in dieser Arbeit beschreiben.

4.2.1 Use-Case-Diagramme

Use-Case-Diagramme beschreiben, was ein System für den Benutzer leistet. Sie werden während Use-Case-Analysen erstellt, bevor mit der Implementierung eines Systems begonnen werden kann [37].

Ein Benutzer wird innerhalb des Diagramms Akteur (*actor*) genannt. Der Akteur benutzt das System durch einzelne Use-Cases. Er stellt dabei Anfragen an das System und erhält Ergebnisse als Ausgabe. Jeder Use-Case ist eine Sammlung von Aktionen, die der Reihe nach abgearbeitet werden müssen, um die Ausgabe zu erzeugen.

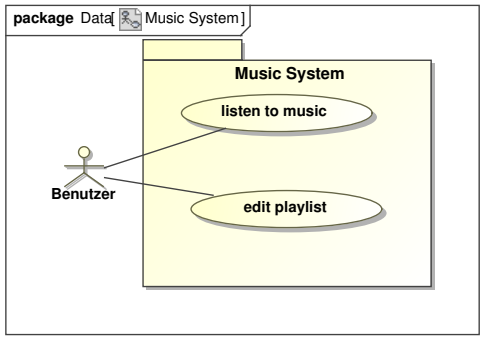


Abbildung 4.3: Use-Case-Diagramm mit zwei Use-Cases.

Abbildung 4.3 zeigt ein Use-Case-Diagramm für den MP3-Player, der zwei Use-Cases enthält. Es gibt nur einen Akteur, den Benutzer. Das System „Music System“ umfasst zwei Use-Cases: Musik hören sowie die Playlist bearbeiten. Der Use-Case zum Musik hören wird im Fallbeispiel umgesetzt. Dieser wird durch andere Diagramme genauer beschrieben, im Beispiel durch ein Aktivitätsdiagramm, das im nächsten Abschnitt vorgestellt wird.

Zwischen Use-Cases können auch Beziehungen bestehen. Ein Use-Case kann einen anderen Use-Case beinhalten oder einen bestehenden erweitern. Komplexere Systeme profitieren von diesen Eigenschaften, indem sie Gemeinsamkeiten von Use-Cases zusammengefassen.

Ein Use-Case kann durch Diagramme, die ein Verhalten beschreiben, detailliert definiert werden. Dafür kommen vor allem Aktivitäts- und Zustandsdiagramme in Frage. In dieser Arbeit werden hierfür stets Aktivitätsdiagramme verwendet. Diese sollen nun näher beschrieben werden.

4.2.2 Aktivitätsdiagramme

Aktivitätsdiagramme (*Activity Charts*) stellen den zeitlichen Ablauf von Tätigkeiten dar. Sie beschreiben Nebenläufigkeit, Ausnahmebehandlung und Trennung der Zuständigkeiten mehrerer Benutzer. Sie erlauben auch, dass andere Aktivitäts- oder Zustandsdiagramme in sie eingebettet werden [136, 141, 138, 142]. Ihre wichtigsten Elemente sind Aktionen (*actions*), verbindende Kanten (*activity edges*) und Kontrollelemente (*control nodes*).

Aktivitätsdiagramme eignen sich besonders, um eine abstraktere Sicht auf Interaktionsabläufe zu beschreiben. Ähnlich wie die Knoten bei den ConcurTaskTrees lässt sich die Aktion in einem Aktivitätsdiagramm als ein Task verstehen. Statt strikt hierarchisch modellieren zu müssen, erlauben Aktivitätsdiagramme sowohl iterative Abläufe als auch Schachtelungen, um Komplexität zu verbergen. Die Schachtelung erlaubt es, bereits modellierte Teile wiederzuverwenden. Für die Schachtelung wird das Element Aktivität (*activity*) eingesetzt. Mit Aktivitäten ist es auch möglich, parametrisierte Knoten zu erstellen, was die Wiederverwendbarkeit fördert.

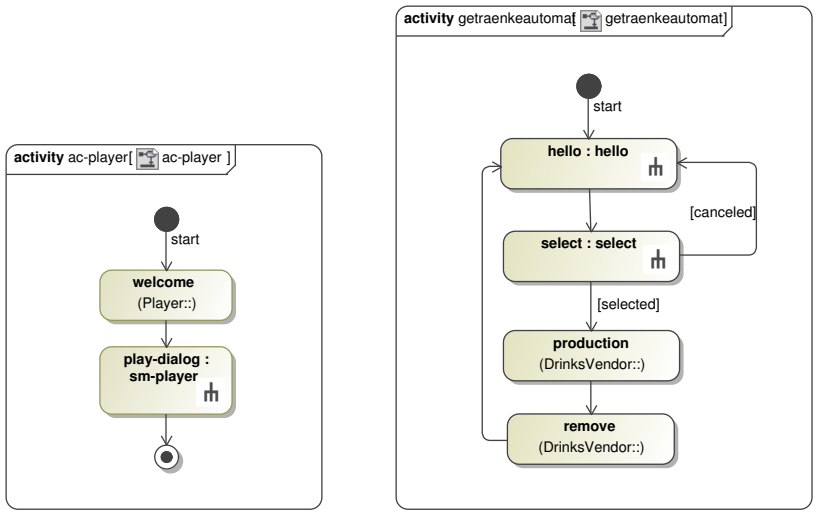
Werden Tasks als Aktionen modelliert, so kann es notwendig sein, einen komplexen Task detaillierter zu beschreiben. Man kann ihn darum als Aktivität beschreiben oder ein Aktivitätsdiagramm in ihn einbetten. Alternativ lässt sich sein Verhalten auch durch ein Zustandsdiagramm beschreiben. Durch Einbettung zusätzlicher Diagramme bleibt eine hierarchische Modellierung weiter möglich.

Die Abbildungen 4.4a und 4.4b zeigen die Aktivitätsdiagramme des MP3-Players und des Getränkeautomaten. Wie man sieht, entspricht ein Aktivitätsdiagramm einem gerichteten Graph. Es enthält Kanten, entlang derer die verschiedenen Aktionen geschehen. Kontrollelemente steuern den Ablauf. Von den Beispielen besitzt nur der Getränkeautomat Verzweigungen. Diese verbinden im Diagramm direkt die verschiedenen Aktionen miteinander. Es ist möglich, statt der direkten Verbindung mit Elementen wie Verzweigungsknoten (*decision nodes*) zu entscheiden, wie im Ablauf die Kanten traversiert werden sollen.

Parallelisierungs- (*Fork nodes*) und Synchronisationsknoten (*join nodes*) können eingesetzt werden, um den Ablauf zu parallelisieren und auch wieder zusammenzuführen. Das Verhalten entlang der Kanten ist ähnlich zu Petrinetzen [153]. Tokens bestimmen, welche Aktionen aktiv sind. An Parallelisierungsknoten wird ein Token vervielfacht, um alle Transitionen zu beschreiten. An Synchronisationsknoten werden die Tokens wieder zusammengefasst.

Zusätzliche Kontrollelemente sind die Start- und Endknoten, die den Ablauf im Graph festlegen. Der Getränkeautomat besitzt keinen Endknoten, da er, einmal eingeschaltet, immer aktiv bleibt. Der Endknoten des MP3-Players wird durch das Ausschalten des Geräts erreicht. Wie man sieht, sind in eine Aktion des MP3-Players und in zwei des Getränkeautomaten weitere Diagramme eingebettet.

Darüber hinaus lassen sich eine Reihe weiterer Elemente verwenden, die in den Beispielen nicht benötigt werden. Unterbrechungsbereiche (*interruptible activity*



(a) Aktivitätsdiagramm MP3-Player (b) Aktivitätsdiagramm Getränkeautomat

Abbildung 4.4: Links das Aktivitätsdiagramm des MP3-Players, rechts das des Getränkeautomaten.

regions) können durch Signale abgebrochen werden. Mit ihrer Hilfe lassen sich beispielsweise Ausnahmeereignisse beschreiben.

Aktivitätsbereiche (*activity partitions*) ermöglichen eine Teilung in verschiedene Bereiche, denen sich Rollen zuordnen lassen. So lassen sich z. B. ein Kunde und ein Sachbearbeiter beschreiben, die gemeinsam an einer Aufgabe arbeiten, aber verschiedene Tätigkeiten ausführen.

Durch Objektknoten (*object nodes*) lassen sich Daten visualisieren, die von einer Aktion für eine spätere Aktion bereitgestellt werden.

4.2.3 Einsatz

Aktivitätsdiagramme werden eingesetzt, um die Aktionen, die ein Benutzer mit einem System erledigen will, genauer zu beschreiben. Jede Aktion sollte dabei in sich abgeschlossen sein. Die Verbindungen werden immer erst nach der Ausführung einer Aktion traversiert, außer es erfolgt eine Unterbrechung.

Aktivitätsdiagramme beschreiben den grob strukturierten Ablauf der Interaktion zwischen Benutzer und System. Will man *direct manipulation* [177] beschreiben, bei der sich die Ausgabe zeitgleich mit der Eingabe ändert, so setzt man innerhalb einer Aktion ein Zustandsdiagramm ein.

Es muss mindestens ein definierter Startpunkt für alle Diagramme existieren, damit sie maschinell verwendbar sind. Dazu wird im Aktivitätsdiagramm auf der obersten Modellierungsebene wenigstens ein Startknoten mit `start` bezeichnet. Der bzw. die Startknoten können dann automatisch ausgewertet werden und die gesamte Anwendung steuern.

Falls eine Aktion durch ein weiteres Diagramm spezifiziert wird, wird dieses gestartet, sobald die Aktion betreten wird. Das aufrufende Diagramm bleibt aktiv, bis das eingebettete Diagramm abgearbeitet ist. Falls in einem aufrufenden Diagramm z. B. eine Unterbrechung stattfindet, werden die betroffenen aufgerufenen Diagramme ebenfalls abgebrochen.

Innerhalb von Aktionen lassen sich Operationen aus Klassendiagrammen verwenden, um Ausgaben vorzunehmen oder den internen Zustand des Systems zu verändern. Die Objekte und ihre Attribute dienen damit als Variablen, die durch Aktionen verändert werden.

Jeder Aktionsknoten muss mit den entsprechenden Ein- und Ausgaben verbunden werden. Hierfür kann ein eindeutiger Name oder eine ID dienen. Wie dies umzusetzen ist, wird im Kapitel zur Realisierung genauer erläutert.

4.3 Dialogmodell

Das Dialogmodell beschreibt, wie sich die Benutzerschnittstelle einer Anwendung verhält. Dies geschieht in der Regel in Zusammenarbeit mit dem Präsentationsmodell, welches die Interaktionselemente dazu bereitstellt (siehe Definition 3). Dazu beschreibt das Dialogmodell die möglichen Aktionen des Benutzers und wie die Anwendung darauf reagieren soll.

In dieser Arbeit werden Zustandsdiagramme eingesetzt, falls ein Dialog nicht bereits ausreichend durch eine Aktion aus einem Aktivitätsdiagramm beschrieben werden kann. Es muss also nicht zwingend ein gesondertes Zustandsdiagramm für jede Aktion geben.

4.3.1 Zustandsdiagramme

Ein Zustandsdiagramm (*statechart diagram*) enthält immer eine Zustandsmaschine (*state machine*), die das Verhalten von Systemen bei bestimmten Ereignissen beschreibt. Zustandsdiagramme entsprechen weitgehend den auf Seite 49 vorgestellten Zustandsdiagrammen von Harel.

Ihre Hauptelemente sind Zustände und Transitionen, welche die Zustände verbinden. Sie besitzen Elemente zur Kontrolle der Ablaufsteuerung, ähnlich zu denen der Aktivitätsdiagramme. Abbildung 4.5 zeigt das Zustandsdiagramm des MP3-Players.

Ein Zustand (*state*) beschreibt den Status in dem sich ein System zu einem bestimmten Zeitpunkt befindet. Er beinhaltet alle aktuellen Attribute eines Systems. Durch eine Transition (*transition*) wird in einen anderen Zustand gewechselt.

Transitionen werden mit der Notation *Trigger[Guard]/Action* geschrieben. Trigger sind Signale, die von außen auf das System einwirken. Sie lösen Transitionen aus. Ein *Guard* ist ein boolscher Ausdruck, der angibt, ob die Transition beschritten werden darf. Es muss darauf geachtet werden, dass sich die Guards gegenseitig ausschließen, denn zu einem Trigger können mehrere Transitionen vorhanden sein. Eine *Action* ruft Operationen auf, um den Systemzustand zu ändern, oder löst weitere Signale aus.

Zustände können einfache oder zusammengesetzte Zustände sein. Einfache Zustände können genauer in ihrem Verhalten beschrieben werden, indem man in ihnen mit *entry*, *do* und *exit* festlegt, was geschehen soll, sobald der Zustand betreten wird, solange im Zustand verweilt wird und sobald er verlassen wird.

Zusammengesetzte Zustände dienen der Hierarchisierung und enthalten in UML Regionen. In Abbildung 4.5 sieht man drei Regionen innerhalb des übergeordneten Zustands `aussen`, die parallel aktiv sind. Sie enthalten Zustandsdiagramme zur Regelung der Lautstärke, der Navigation in der Playlist und zur Steuerung der Wiedergabe.

Es gibt verschiedene Arten von Signalen, welche Transitionen auslösen. Ein *SignalTrigger* wird erzeugt, wenn ein bestimmtes Signal anliegt. Für ein Zustandsdiagramm, das das interne Verhalten eines Systems beschreibt, benutzt man vor allem diesen Triggertyp. Jedes Mal, wenn sich der Wert einer Variablen ändert, wird der Guard neu geprüft. Sollte sich sein Wert auf *wahr* ändern, wird ein ChangeTrigger erzeugt, der als Auslöser der Transition dient. Ein *ChangeTrigger* ist immer auch implizit gegeben, wenn ein Guard ohne Trigger verwendet wird. *TimeTrigger* werden nach Ablauf einer Zeitspanne ab Betreten des Quellzustands oder zu einem fest gegebenen Zeitpunkt erzeugt.

Zustandsdiagramme bieten noch eine Reihe weiterer Elemente, wie die Historie (*history*), auf die an dieser Stelle nicht weiter eingegangen wird.

4.3.2 Einsatz

In dieser Arbeit werden Zustandsdiagramme dazu eingesetzt, das Verhalten innerhalb von Aktionen aus Aktivitätsdiagrammen genauer zu beschreiben, sofern dies notwenig ist. Wann welches Diagramm eingesetzt wird, ist eine Design-Entscheidung. Falls sich eine Interaktion mit Zuständen besser beschreiben lässt,

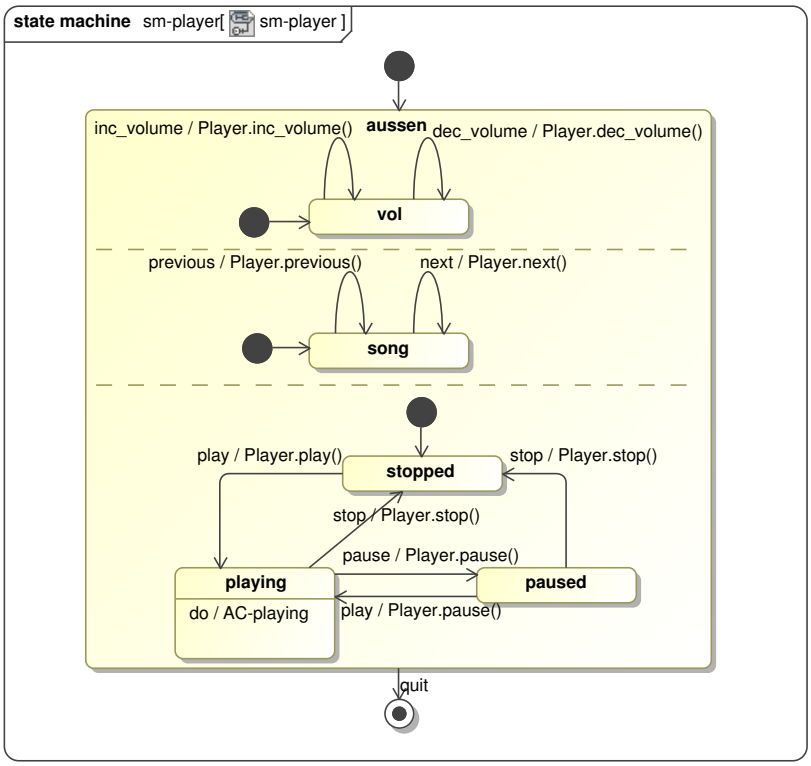


Abbildung 4.5: Zustandsdiagramm für den MP3-Player. Der Oberzustand enthält drei Regionen mit parallel aktiven Zustandsdiagrammen.

wird das Zustandsdiagramm eingesetzt. Viele Interaktionen lassen sich gut durch Zustände beschreiben, beispielsweise die Wahl eines Getränks an einem Automaten oder die Wiedergabe von Musik.

Entlang der Transitionen können durch die Actions Operationen auf Objekten ausgeführt werden. Durch sie lassen sich, wie bei Aktivitätsdiagrammen, Ausgaben vornehmen oder der interne Zustand des Systems verändern. Eingaben des Benutzers geschehen durch Trigger, welche die Transitionen schalten. Durch Guards lassen sich bei Verzweigungen die richtigen Transitionen auswählen.

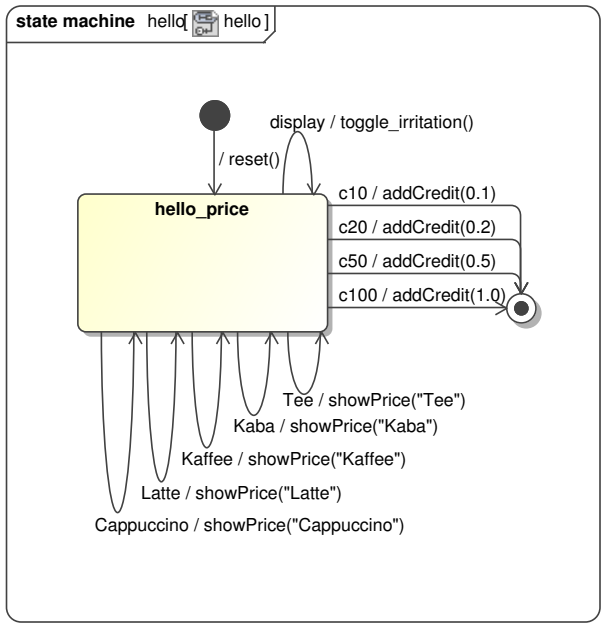


Abbildung 4.6: Zustandsdiagramm des Getränkeautomaten mit Begrüßung und Geldeinwurf.

In Abbildung 4.6 und 4.7 sind die Zustandsdiagramme des Getränkeautomaten zu sehen. Die Begrüßung wird durch einen einzelnen Zustand umgesetzt. Im Diagramm in Abbildung 4.7 ist die Auswahl eines Getränks zu sehen, die mit einem zusammengesetzten Zustand arbeitet. An den Transitionen stehen die verschiedenen Signale für Benutzereingaben und die Operationen, die aufgerufen werden.

Zustandsdiagramme können andere Diagramme, die ein Verhalten beschreiben, aufrufen. Dies sind Zustands- und Aktivitätsdiagramme. Abbildung 4.8 zeigt ein Aktivitätsdiagramm, das in den Zustand `playing` des Zustandsdiagramms in Abbildung 4.5 eingebettet ist. Das Aktivitätsdiagramm, das den Datenfluss und die Umwandlung der Daten während des Abspielens beschreibt, wird durch die Anweisung `do/AC-playing` aufgerufen.

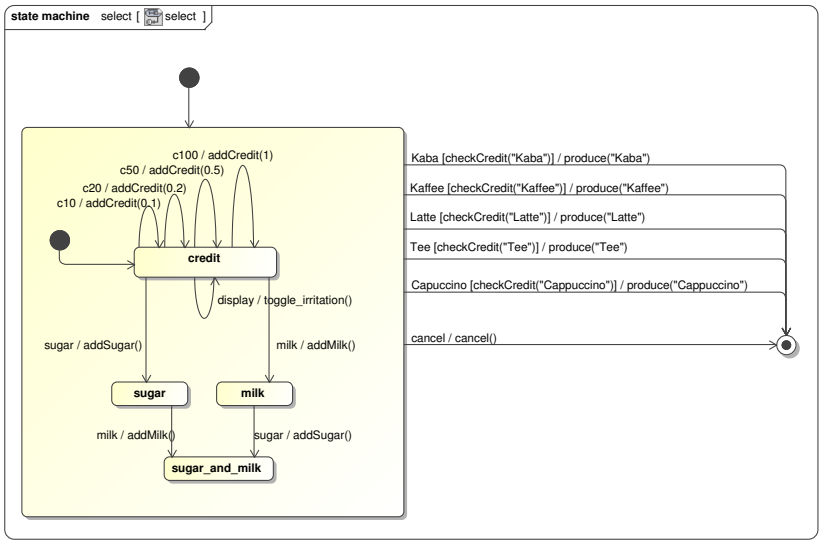


Abbildung 4.7: Zustandsdiagramm des Getränkeautomaten für die Auswahl eines Getränks.

4.4 Ergänzende Diagramme

Es gibt noch weitere Diagramme, die im Entwicklungsprozess eingesetzt werden können. Diese sind wie das Use-Case-Diagramm nicht zwingend nötig, um eine Interaktionsanwendung zu erstellen, unterstützen aber die Arbeit. Eine ausgewählte Menge wird im Folgenden kurz vorgestellt.

Paketdiagramme helfen bei der Entwicklung größerer Systeme und beschreiben, wie Sammlungen von Klassen zusammengefasst werden. Durch sie kann die Übersichtlichkeit besser gewahrt werden, da sie eine Gliederung nach funktionalen Aspekten vornehmen. Im Beispiel des MP3-Players kann man sich Pakete für das Abspielen von Musik oder auch für unterschiedliche Playlist-Navigation für verschiedene Varianten des Produkts vorstellen.

Gerade bei größeren Systemen kommen auch Komponentendiagramme zum Einsatz. Sie beschreiben die Struktur eines Systems zur Laufzeit. Insbesondere verdeutlichen diese Diagramme, wie die verschiedenen Komponenten erzeugt werden und was für Verbindungen zwischen ihnen bestehen. Diese Eigenschaften sind z. B. für die Spezifikation von Schnittstellen nützlich.

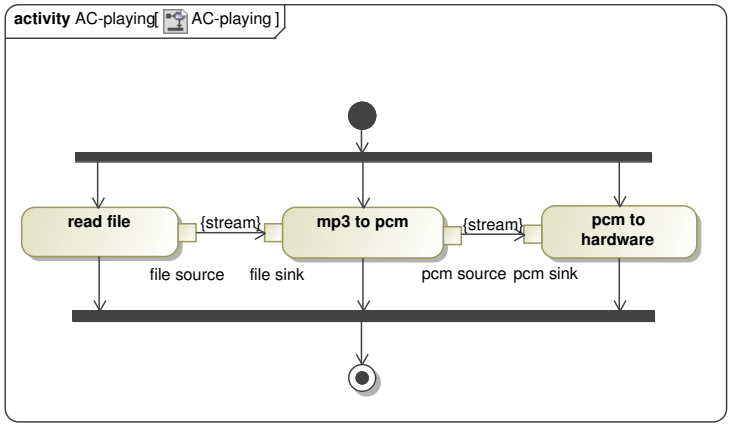


Abbildung 4.8: Aktivitätsdiagramm für das Wandeln von MP3 zu PCM.

Sequenzdiagramme zeigen den Austausch von Nachrichten zwischen Kommunikationspartnern innerhalb des Systems. Sie beschreiben dabei die zeitliche und logische Abfolge von Nachrichten. Sie können z. B. aufführen, in welcher Reihenfolge Methoden von Objekten innerhalb des Getränkeautomaten aufgerufen werden können.

4.5 Präsentationsmodell

Nachdem vorgestellt wurde, wie die verschiedenen Modelle zur abstrakten Beschreibung der Interaktion und der Domäne mit UML umgesetzt werden, wird als nächstes auf das Präsentationsmodell und die Kontextmodelle eingegangen.

Das Präsentationsmodell repräsentiert die verschiedenen Elemente der Benutzerschnittstelle und ihre Ausprägungen in Modalitäten. Das Präsentationsmodell wird nicht durch eine Diagrammart der UML dargestellt. Es beinhaltet eine konkrete Ausprägung einer Benutzerschnittstelle und wird aus anderen Modellen abgeleitet. Zu einer Ableitung werden in der Regel die drei Kontextmodelle (Benutzer-, Umgebungs- und Gerätemodell) berücksichtigt. Alle diese Modelle sind nicht direkt in UML vorhanden.

In der Ausprägung als *concrete* (CUI) oder *final user interface* (FUI) hat das Präsentationsmodell einzelne Ausprägungen, die Benutzerschnittstellen für eine einzelne Modalität auf einer spezifischen Plattform mit Anpassung an den Benutzer entsprechen. Für diese Arbeit spielt es keine Rolle, ob dabei binärer Code für ein

FUI entstanden ist oder die Vorstufe eines CUI erstellt wurde. Der Unterschied ist lediglich die Übersetzung vom *concrete* in das *final user interface*. Daher beschränkt sich die folgende Betrachtung auf CUIs.

Es existieren zwei Möglichkeiten, um die CUIs zu erhalten. Eine Möglichkeit ist, die CUIs aus den anderen Modellen unter Einbeziehung der Kontextmodelle zu generieren. Die andere Möglichkeit ist, falls es nur wenige Kontexte gibt, die Benutzerschnittstellen händisch zu erzeugen. Bevor beide Möglichkeiten beschrieben werden, muss noch das Mapping zwischen den abstrakten Interaktionsmodellen und dem Präsentationsmodell erstellt werden.

4.5.1 Mapping zwischen Aufgaben-, Dialog- und Präsentationsmodell

Um festzustellen, zwischen welchen Elementen Mappings notwendig sind, betrachtet man die Verbindungen des Präsentationsmodells mit anderen Modellen. Durch das Aufgaben- und das Dialogmodell wird beschrieben, welche Teile des Präsentationsmodells wann aktiv sein sollen und wie sie sich verhalten soll. Das Domänenmodell ist indirekt angebunden: Operationen auf seinen Objekten können via Dialog- und Aufgabenmodell durchgeführt werden. Beim Ansatz dieser Arbeit werden die Elemente von Aktivitäts- und Zustandsdiagrammen mit dem Präsentationsmodell verbunden. Tabelle 4.1 stellt einen Überblick über alle verknüpfbaren Elemente dar.

Diagramm	Element
Aktivitätsdiagramm	Aktion Operation innerhalb einer Aktion
Zustandsdiagramm	Zustand Trigger Actions, die durch Trigger ausgelöst werden

Tabelle 4.1: Verknüpfbare Elemente von Aktivitäts- und Zustandsdiagrammen.

Im Aktivitätsdiagramm können der Name bzw. die ID einer Aktion oder die in einer Aktion spezifizierten Operationen verknüpft werden. Da Aktionen abgeschlossene Aufgaben repräsentieren, eignet sich ein Mapping zu Präsentationselementen, die einen kompletten Teildialog beschreiben. Dies können z. B. einfache Eingaben sein, die in sich eine abgeschlossene Aufgabe bilden. Ein anderes Beispiel sind Ausgaben, mit denen eine Anwendung ihren Benutzer informiert.

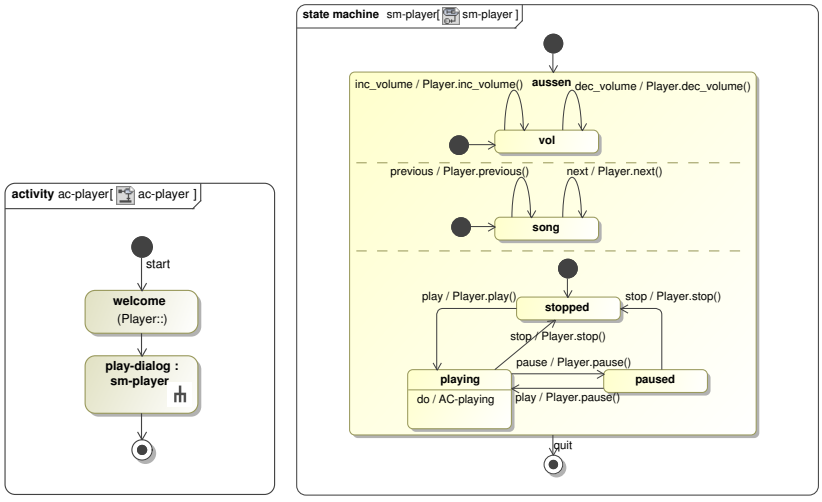
Bei Zustandsdiagrammen können ebenfalls der Name bzw. die ID eines Zustands für ein Mapping verwendet werden. Darüber hinaus sind die Trigger an den Transitionen die wichtigsten Elemente. Sie entsprechen Interaktionen des Benutzers und werden mit den passenden Interaktionselementen verbunden. Dabei sind

auch die Actions, die durch Trigger ausgelöst werden können, zu berücksichtigen. Sie dürfen Signale senden, die auch zur Kommunikation mit dem Benutzer genutzt werden können.

Die Menge der betroffenen Elemente aus den abstrakten Interaktionsmodellen ist also überschaubar. Es stellt sich noch die Frage, wie ein Mapping umgesetzt wird und welche Elemente in einem Präsentationsmodell betroffen sind.

Am einfachsten lassen sich die Elemente der Interaktionsmodelle mit der Präsentation verbinden, in dem ein *model linking* über die Namen der Signale, Zustände, Aktionen und Ausgabeereignisse vorgenommen wird. Eine Verbindung über den Namen allein stellt eine Verbindung dar, die mit moderatem Aufwand pflegbar ist[2].

Eine solche Verbindung lässt sich durch den Einsatz von Events realisieren. Alle Trigger bei Zustandsdiagrammen bezeichnen bereits Events. Von den anderen Elementen dienen einige als Eventquellen, so z. B. das Betreten eines Zustands oder einer Aktion. Auf Seiten des Präsentationsmodells müssen diese Events abgefangen und entsprechend verarbeitet werden.



(a) Aktivitätsdiagramm MP3-Player (b) Zustandsdiagramm MP3-Player

Abbildung 4.9: Links das Aktivitätsdiagramm des MP3-Players, rechts das Zustandsdiagramm der Aktion `play-dialog`.

[2]Meilir Page-Jones beschreibt dies im Detail [145].

Das Beispiel des MP3-Players soll zur Verdeutlichung des Mappings dienen. In Abbildung 4.9a und 4.9b sieht man noch einmal sein Aktivitäts und Zustandsdiagramm. Wird der Player gestartet, so wird die Aktion `welcome` betreten. Diese löst einen Event an das Präsentationsmodell aus und kann dort z. B. eine Begrüßungsnachricht ausgeben. Mit dem Betreten der Aktion `play-dialog` wird ebenfalls ein Event ausgesendet. Nimmt man eine GUI als Modalität an, wird bei dieser ein Fenster mit dem gleichen Namen angezeigt, das die Bedienelemente für die Anwendung enthält. Durch das Betreten des Zustandsdiagramms und seiner Unterzustände werden weitere Events an das Präsentationsmodell gesendet. Diese können in diesem Beispiel ignoriert werden, da sie keine neuen Informationen für den Benutzer liefern. Betätigt der Benutzer eine Schaltfläche um Musik abzuspielen, so löst diese das Ereignis `play` aus, das als Trigger im Zustandsübergang die Transition zu `playing` aktiviert. Dabei wird die Operation `play()` ausgeführt, welche die Wiedergabe der Musik startet.

Damit das Mapping zwischen den Modellen zuverlässig funktioniert, muss sichergestellt werden, dass für alle relevanten Events bei allen beteiligten Modellen Vorkehrungen zur Bearbeitung getroffen worden sind. Außerdem müssen die verwendeten Namen eindeutig sein. Wie ein Verfahren zur automatischen Erzeugung eines solchen Mappings aussehen kann, wird im nächsten Abschnitt beschrieben.

4.5.2 Generierung von Benutzerschnittstellen

Auch wenn es nicht das Ziel dieser Arbeit ist, Verfahren zur Generierung von Benutzerschnittstellen zu untersuchen sondern die Modelle zu interpretieren, lohnt es sich, auf diese verwandte Thematik einzugehen. Im Folgenden wird eine Skizze der notwendigen Techniken präsentiert.

Um Benutzerschnittstellen innerhalb eines Präsentationsmodells zu generieren ,genügt es, *concrete user interfaces* (CUIs) zu schaffen, wie in Abschnitt 4.5 festgestellt wurde. Für jede Modalität ist dabei ein eigenes CUI zu erstellen.

Die Ableitung eines Modells aus anderen Modellen entspricht dem Mapping *model derivation*. Jedes CUI ist von den Kontextmodellen abhängig. In ein CUI fließen Informationen aus Benutzer-, Umgebungs- und Gerätemodell ein. Es ist notwendig, Regeln für die Ableitung aufzustellen, die die verschiedenen Modelle als Ausgangspunkt verwenden. Eine Reihe von Attributen der Kontextmodelle ist statisch, andere Attribute ändern sich zur Laufzeit. Statische Attribute sind z. B. der Name des Benutzers oder die Auflösung des verwendeten Geräts. Dynamische Attribute sind dagegen Lautstärke, Helligkeit oder die Aufmerksamkeit des Benutzers in seiner Umgebung.

Zur Erstellung von Regeln für Transformationen existieren diverse Arbeiten. Stolze et al. stellten in ihrer Arbeit fest, dass der Einsatz von Modellen zur Erstellung von Benutzerschnittstellen die Entwicklung deutlich beschleunigt [184]. Arbeiten zu Transformationen gibt es schon seit einiger Zeit, Wiecha und Boies stellten

1990 einen solchen Ansatz vor, der zusätzlich Stilinformationen berücksichtigte [206]. Ein neuerer Ansatz stammt von da Silva et al., die Regeln für die automatische Generierung von GUIs einsetzten [179]. Die Regeln zur Transformation der Interaktionsmodelle benötigen primär die wenigen im letzten Abschnitt vorgestellten Elemente, d. h. Trigger, Operationen und die Namen von Aktionen und Zuständen.

Im Lauf der Zeit wurden verschiedene spezielle Sprachen für Transformationsregeln entwickelt. Ursprünglich wurden deklarative Sprachen oder Constraints eingesetzt [63, 169]. T:XML ist eine XML-basierte Transformationssprache, die von López-Jaquero et al. 2009 vorgestellt wurde [101]. Besonders auf das Metamodell von UML ist dagegen *MOF Queries, Views, Transformations* (MOF QVT) abgestimmt, eine Transformationssprache, die von der Object Management Group entwickelt wird [140].

Regeln lassen sich in verschiedene Kategorien gliedern. Neben den eigentlichen Transformationsregeln existieren auch Meta-Regeln, die die Anwendbarkeit von Regelmengen auf einen Kontext überprüfen. Eine besondere Sorte Regeln sind Ausnahmen, die von Entwicklern für Sonderfälle gewünscht werden. Dies kann z. B. durch Besonderheiten im Design eines Endgeräts oder durch erwünschtes Branding für ein Produkt notwendig werden. Pederiva et al. nennen diese Regeln *Beautifications* und beschrieben ein Verfahren, wie man Anpassungen an Standardtransformationen wieder in den Prozess zurück fließen lassen kann [151]. Eine ausführlichere Behandlung der Thematik von Regeln und Generierung erfolgt in der Dissertation von Mark Poguntke, die aktuell in Bearbeitung ist [156].

Trotz des Vorhandenseins von Regeln und jahrelanger Forschung im Bereich der automatischen Transformation von Modellen, sollte nicht vergessen werden, dass die Generierung ein komplexes Thema ist. Caffiau et al. beschrieben in ihrer Arbeit einige auftretende Probleme [22]. Aufgabenmodelle sind weitgehend verschieden von Präsentationsmodellen. Die Inhalte eines Dialogs müssen sich nicht genau mit einem Task decken. Ein Task kann in mehrere Teildialoge aufgeteilt werden oder mit einem anderen Task in einem Dialog kombiniert repräsentiert werden. Benutzerfehler werden in der Regel in Aufgabenmodellen ignoriert, schließlich ist die Auswahl der richtigen Interaktionselemente nicht trivial. Die Wahl zwischen einem Dropdown-Menü und Feldern mit Radiobuttons in einer GUI kann nicht nur von der Displaygröße des Ausgabegeräts abhängen. Gajos et al. ergänzten, dass der Generierungsprozess zu vom Benutzer vorhersehbaren Ergebnissen führen muss, da sonst eine Verunsicherung des Benutzers bei der Bedienung eintreten kann [54].

Nachdem die notwendigen Techniken zur Generierung von Benutzerschittstellen skizziert wurden und verschiedene Aspekte kritisch betrachtet wurden, soll die zweite Variante, das händische Erzeugen von Benutzerschnittstellen, kurz beschrieben werden.

4.5.3 Direktes Mapping zu Modalitäten

Zur Demonstration des in diesem Kapitel vorgestellten Verfahrens genügt ein Mapping, das von Hand hergestellt wird. Das Vorgehen besteht im wesentlichen aus drei Schritten:

1. Auswahl einer Modalität,
2. Finden aller relevanten Elemente in den Diagrammen,
3. Integration der Namen der Elemente in die Benutzerschnittstelle.

Diese Schritte werden am Beispiel des MP3-Players erläutert.

Zuerst wird eine Modalität ausgewählt, die verwendet werden soll. Für diese wird dabei festgelegt, welche Geräte eingesetzt werden. Im Fall des MP3-Players soll eine GUI für einen Desktop-Rechner erstellt werden. Als Beschreibungssprache für die GUI soll GTK verwendet werden [198]. Damit nicht alle Elemente in Code angelegt werden müssen, wird Glade eingesetzt, eine Sprache, die GTK-Widgets in XML beschreibt [199].

Um alle relevanten Elemente zu finden, wird auf Tabelle 4.1 aus Abschnitt 4.5.1 zurückgegriffen. Im Fall des MP3-Players sind die Elemente innerhalb des Zustandsdiagramms zwei Aktionen, sechs Zustände und zehn Transitionen mit Triggern (vergleiche Abbildung 4.9a und 4.9b).

Beim Erstellen der GUI werden die Namen der Aktionen, Zustände und Trigger in die Oberfläche integriert. Buttons werden mit Sendeereignissen für Events mit den Namen der Trigger versehen. Nicht alle Aktions- und Zustandsnamen müssen berücksichtigt werden. In diesem Beispiel genügt es, ein Dialogfenster für die Aktion `welcome` und ein weiteres für `play-dialog` anzulegen. Abbildung 4.10 zeigt die Oberfläche des Glade User Interface Designers. Auf der rechten Seite sieht man im Kontextmenü zum ausgewählten Button, wie dieser mit einem Event verknüpft wird.

4.6 Fazit

In diesem Kapitel wurde beschrieben, wie mit UML Interaktionsanwendungen modelliert werden können. Für das Domänenmodell wurde der Einsatz von Klassen beschrieben. Es wurde ein Verfahren vorgestellt, bei dem Aktivitätsdiagramme die grobe Strukturierung übernehmen und Zustandsdiagramme für *direct manipulation* oder zustandsbehaftete Situationen eingesetzt werden. Zusätzlich wurden Use-Case-Diagramme vorgestellt und kurz über den Einsatz von Paket-, Komponenten- und Sequenzdiagrammen berichtet.

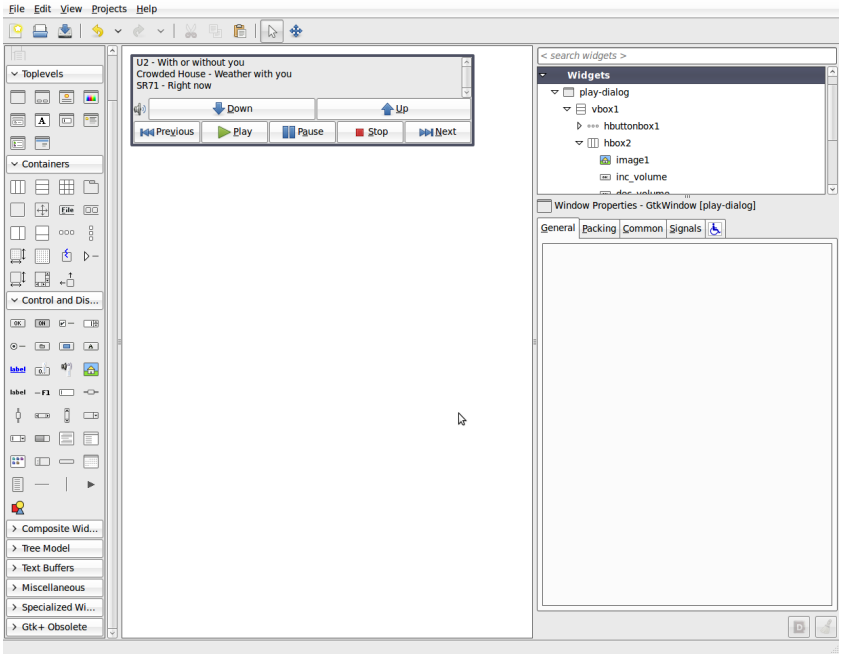


Abbildung 4.10: Glade User Interface Designer, rechts im Kontextmenü die Verknüpfung eines Buttons mit einem Event.

Der vorgestellte Ansatz enthält weiterhin die Möglichkeit zur hierarchischen Modellierung, er bietet jedoch eine Reihe von zusätzlichen, UML-spezifischen Elementen. Eine hierarchische Aufgabenanalyse wie bei den ConcurTaskTrees bleibt damit weiterhin möglich. Aufgaben- und Dialogmodell sind durch die Kopplung der Diagramme in UML stärker als bisher miteinander verbunden. Ein besonderer Vorteil des Ansatzes ist, dass die UML eine weit verbreitete Sprache ist. Eine Ableitung von Präsentationsmodellen ist möglich und wie Abschnitt 4.5.1 zeigt, sind nur wenige Elemente für Mappings zu betrachten. Es genügt allerdings nicht, lediglich das Mapping über die Namen der Elemente zwischen den Modellen herzustellen. Auch die zu Grunde liegende Architektur muss diesen Ansatz unterstützen und die Verteilung der Events vornehmen.

Das Mapping Problem kann durch keinen Ansatz vermieden werden. Es lassen sich jedoch für die gesamte UML vorhandene Mappings einsetzen. Dies reduziert den Aufwand, Mappings zu erstellen und zu pflegen, erheblich.

Auf das Evaluationsmodell wurde in diesem Kapitel nicht eingegangen. Evaluationsmodelle lassen sich jedoch unabhängig zum hier vorgestellten Verfahren einsetzen. Die in UML erstellten Modelle lassen sich auch bedingt zur Evaluation verwenden. Bei einer Task-Analyse besteht z. B. die Möglichkeit auf die Aktionen im Aktivitätsdiagramm zurückzugreifen.

5 Architekturen für multimodale Benutzerschnittstellen

> Architektur ist Harmonie und Einklang aller Teile, die so erreicht wird, dass nichts weggenommen, zugefügt oder verändert werden könnte, ohne das Ganze zu zerstören.
>
> *Leon Battista Alberti, 1452*

In Kapitel 4 wurde vorgestellt, wie mit Hilfe von UML interaktive Anwendungen modelliert werden können. Dieses Kapitel beschreibt eine geeignete Software-Architektur, die multimodale Anwendungen realisierbar macht. Unter dem Begriff Software-Architektur wird in dieser Arbeit eine Strukturierung in Komponenten, die einen Referenzrahmen für Implementierungen vorgibt, verstanden. Um eine solche Architektur zu finden, werden zuerst bestehende Ansätze für Software-Architekturen bewertet und anschließend eine geeignete Architektur beschrieben, welche den Anforderungen modellbasierter Interaktionsanwendungen genügt.

Die bisher vorgestellten Modelle ermöglichen die Gliederung einer Architektur in Komponenten, die wiederum durch konzeptionelle Modelle beschrieben werden können. Das Zusammenspiel zwischen den einzelnen Modellen wird auf die Verbindung der Komponenten übertragen.

Wenn mehrere Modalitäten gleichzeitig eingesetzt werden, so kann es zu Konflikten bei der Benutzereingabe kommen. Ein Benutzer kann beispielsweise über verschiedene Modalitäten widersprüchliche Befehle eingeben. Darum müssen die Eingaben aufbereitet und zu einem einzigen Eingabestrom verschmolzen werden. Immer, wenn der Zustand des Dialogs sich verändert, muss diese Änderung an alle eingesetzten Modalitäten kommuniziert werden. Die Modalitäten können durch diese Information ein Update ihrer Ausgaben an den Benutzer

vornehmen. In diesem Kapitel wird deshalb auf die Fusion der Eingabe und die Fission der Ausgabe an die Modalitäten gesondert eingegangen.

Anforderungen an Architekturen

Im Lauf der Zeit entstanden eine Reihe verschiedenster Software-Architekturen für Benutzerschnittstellen. Nicht alle waren von Anfang an für multimodale Systeme ausgelegt. Ein Teil der bestehenden Architekturen lässt sich so erweitern, dass er für multimodale Systeme nutzbar wird. Von besonderem Interesse sind jedoch Architekturen, die multimodale Systeme direkt unterstützen.

Bevor die Ansätze untersucht werden können, ist auf Anforderungen an Architekturen einzugehen. In Abschnitt 3.2.1 wurden bei den Anforderungen an Notationen bereits eine Reihe von Eigenschaften beschrieben, die ebenfalls durch die hier vorgestellten Architekturen unterstützt werden sollten.

Eine Architektur für modellbasierte Interaktionsanwendungen sollte die erstellten Modelle einsetzen, um mit ihnen die Interaktion mit dem Benutzer zu steuern. Sie sollte ferner dynamisch um Modalitäten erweiterbar sein. Eine solche Architektur sollte vorzugsweise verschiedene Plattformen zur Laufzeit unterstützen. Dies ist dann wichtig, wenn mehrere Modalitäten verwendet werden sollen, die an jeweils eigene spezielle Plattformen gebunden sind.

5.1 Ansätze für Architekturen

Im folgenden werden verschiedene Ansätze für Architekturen vorgestellt, die modellbasierte Interaktionsanwendungen unterstützen. Eine gesonderte Kategorie stellen Toolkits dar, auf sie wird als erstes eingegangen.

Bestehende Toolkits sind die am stärksten verbreitete Art, Benutzerschnittstellen für Systeme zu entwickeln. Sie erfüllen eine Reihe der Anforderungen. Sie unterstützen eine plattformunabhängige Modellierung, sind internationalisierbar und durch *Themes* personalisierbar. Allerdings erfüllen sie einige notwendige Anforderungen nicht. Sie unterstützen nicht die parallele Nutzung mehrerer Modalitäten und sie bieten auch nicht die Möglichkeit, eine Anwendung parallel auf mehreren Plattformen verfügbar zu machen. Hierfür sind umfassendere Ansätze notwendig.

5.1.1 User Interface Management Systeme (UIMS)

Der Ausdruck *User Interface Management System* (UIMS) wurde 1982 von Kasik eingeführt, nachdem er untersucht hatte, wie grafische Oberflächen das Gebiet der Mensch-Maschine Interaktion erweitern würden [87].

Ziele eines User Interface Management Systems sind, auf einer abstrakten Ebene Eingabe und Ausgabe zu spezifizieren und dem Benutzer standardisierte Benutzerschnittstellen anzubieten. Sie trennen die Anwendung von der Benutzerschnittstelle und beschreiben diese auf einer abstrakteren Ebene als Toolkits. UIMSe unterstützen plattformunabhängige Systeme und können parallel mehrere Benutzerschnittstellen bereitstellen. Damit eignen sie sich für multimodale Systeme.

Die wichtigsten Bestandteile eines UIMS sind Komponenten für die Präsentation der Benutzerschnittstelle, eine Steuerung der Interaktion sowie eine Schnittstelle zur Anwendung [52]. Diese Komponenten finden sich z. B. in *Garnet*, einem UIMS das zwischen 1987 und 1994 an der Carnegie Mellon Universität entwickelt wurde [193, 118]. Das System wurde für Common Lisp und X11 sowie Macintosh entwickelt und unterstützt den Anwender bei der Erstellung von grafischen Oberflächen durch einen *Rapid Application Development*-Ansatz. Garnet konzentriert sich auf grafische Oberflächen, Echtzeit-Fähigkeit von Anwendungen und Editoren für Oberflächen.

Die Aufteilung in Anwendung, Dialogsteuerung und Präsentation ist ein grundlegender Beitrag der User Interface Management Systeme. Da es sich um einen rein konzeptuellen Ansatz handelt, werden Implementierungsdetails der Komponenten in den Arbeiten von Kasik und Foley nicht ausgeführt [87, 52]. Es existieren keine aktuell einsetzbaren UIMSe mehr. Eine aktuelle Arbeit stammt aus dem Jahr 2004. Leland et al. stellten ein Konzept für ein UIMS für TUIs vor, das bisher jedoch nicht implementiert wurde [96].

5.1.2 Seeheim-Modell

Das Seeheim-Modell entstand kurz nach der Einführung von User Interface Management Systemen. Es wurde 1985 bei einem Workshop in Seeheim, bei Darmstadt, vorgestellt und besteht aus drei Komponenten, welche ein Bindeglied zwischen Benutzer und Anwendung darstellen [154]: Eine Schnittstelle zur Anwendung, eine Dialogsteuerung und eine Präsentationskomponente.

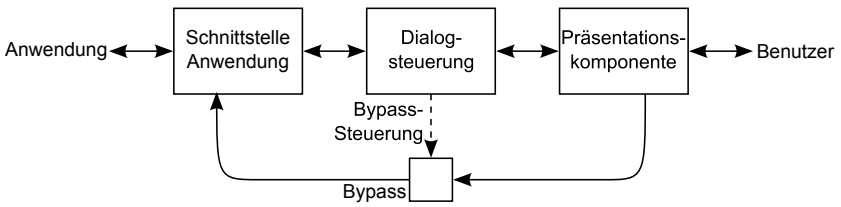


Abbildung 5.1: Das Seeheim-Modell.

Abbildung 5.1 stellt die Komponenten des Seeheim-Modells grafisch dar. Die Schnittstelle zur Anwendung beschreibt den Zugriff auf die Semantik der jeweiligen Anwendung. Durch sie erfolgt auch die Kommunikation mit der Anwendung. Die Dialogsteuerung realisiert die Interaktivität der Anwendung. Sie verarbeitet Benutzereingaben und bestimmt den Verlauf der Interaktion zwischen Benutzer und System. Die Präsentationskomponente steuert die Erscheinung der Benutzerschnittstelle, inklusive welche Ein- und Ausgabemöglichkeiten dem Benutzer zur Verfügung stehen. Zusätzlich existiert noch eine Komponente, die als *Bypass* vorgesehen ist.

Durch sie wird eine schnelle semantische Rückmeldung ermöglicht. Solch eine Rückmeldung wurde eingeplant, um in Fällen wie beispielsweise dem Zeichnen von Linien die Eingabe zur Anwendung zu beschleunigen. Ohne die Umgehung müssten alle Eingabeereignisse in der Dialogsteuerung verarbeitet werden, bevor sie an die Anwendung weitergeleitet würden. Aufgrund der geringeren Leistungsfähigkeit der damaligen Hardware bedeutete der Bypass ein schnelles Ansprechverhalten eines Systems. Durch die Bypass-Steuerung kann die Dialogsteuerung bestimmen, für welche Interaktionen der Bypass verwendet werden soll. In der Abbildung ist die Steuerung mit einem gestrichelten Pfeil markiert.

Das Seeheim-Modell ist eine Konkretisierung des UIMS-Ansatzes. Es beinhaltet weiterhin die Dreiteilung der Komponenten, trifft aber keine Aussagen über die Anbindung von mehreren Modalitäten.

5.1.3 Arch und Arch-Slinky Modell

Das von Bass 1992 vorgestellte Modell ist eine Weiterentwicklung des Seeheim-Modells und vereinigt viele Vorzüge der bisher vorgestellten Ansätze [12]. Die Komponenten werden noch ausführlicher spezifiziert und ihre Anzahl auf fünf erhöht. Es handelt sich um zwei Modelle, das Arch-Modell und das Slinky-Metamodell, und die möglichen Variationen durch das Metamodell.

Abbildung 5.2 illustriert. woher das Arch-Modell seinen Namen hat. Die Komponenten des Modells sind in einem Bogen angeordnet. Die Dialogsteuerung existiert weiter als *Dialogue Component*. Die bisherige Präsentationskomponente wird in eine *Presentation Component* und in eine *Interaction Toolkit Component* aufgeteilt. Die *Presentation Component* ist eine Adapterkomponente, die Toolkitunabhängig arbeitet. Die *Interaction Toolkit Component* realisiert die Benutzerschnittstelle. Die Schnittstelle zur Anwendung wurde um die *Domain Adaptor-Component* erweitert. Bei ihr handelt es sich um eine Vermittlungskomponente, die vom Benutzer benötigte Funktionalität bereitstellt, welche nicht direkt vom System angeboten wird. Dies können z. B. Kombinationen von mehreren Methoden des Systems oder zusätzlich implementierte Methoden sein. Die *Domain-Specific Component* enthält die Domänen-Objekte und damit die Anwendung. Zwischen den verschiedenen Komponenten wird durch unterschiedliche Objekte kommuniziert, die an den Pfeilen annotiert sind.

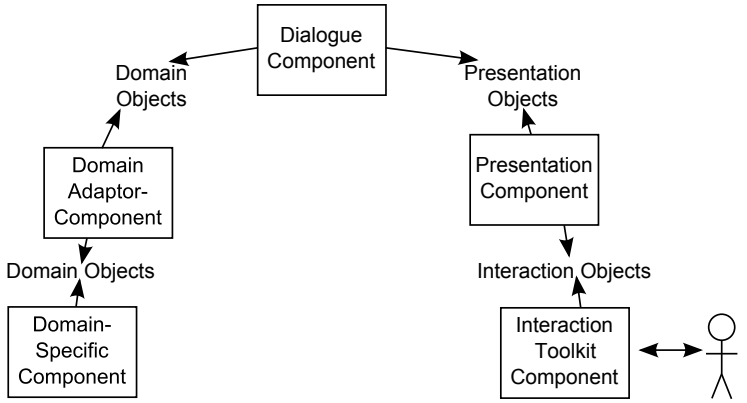


Abbildung 5.2: Das Arch Modell nach [12].

Abbildung 5.3 zeigt das Slinky Metamodell. Es zeigt, dass die einzelnen Komponenten des Arch Modells in unterschiedlicher Komplexität realisiert werden können.

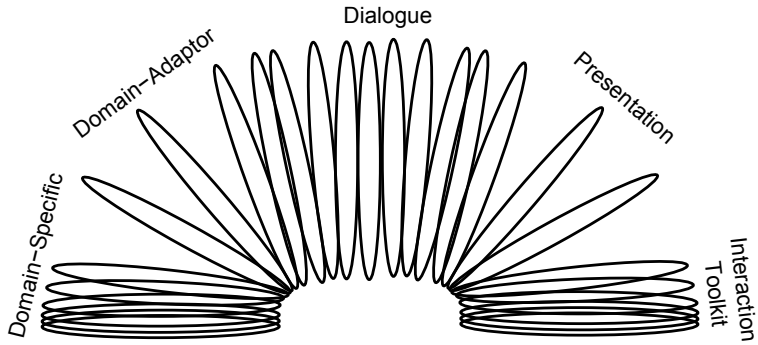


Abbildung 5.3: Das Slinky Metamodell nach [12].

Das Arch-Slinky Modell ist eine Kombination aus den Komponenten des Arch-Modells und dem Slinky-Metamodell. Das Arch-Slinky Modell trägt der Praxis Rechnung, in der die Komponenten des Arch-Modells in unterschiedlichen Ausprägungen realisiert werden. Das Modell spezifiziert detailliert die Komponenten einer Interaktionsanwendung und trifft Aussagen über die Zuordnung von Funktionalität. Durch die Definition der Kommunikationsobjekte lässt sich eine platt-

formübergreifende Architektur verwirklichen. Es trifft keine Aussagen darüber, wie die Dialogsteuerung realisiert wird und wie mehrere Modalitäten angebunden werden können.

5.1.4 Multimodale Präsentationskomponenten

Die bisherigen Ansätze schließen zwar multimodale Interaktion nicht aus, beschreiben aber auch nicht, wie diese im Detail realisiert werden kann. Im selben Zeitraum wie die oben beschriebenen Modelle entstanden andere Ansätze, die hierauf eingehen. Die zwei bekanntesten sind *Model View Controller* und *Presentation Abstraction Control*.

Das *Model View Controller* (MVC) Konzept wurde 1979 von Reenskaug beim Xerox Palo Alto Research Center (PARC) entwickelt [162]. Die erste Implementierung wurde für die Erstellung von Benutzeroberflächen mit Smalltalk-80 verwendet [95]. Ziel war die Trennung von Benutzerschnittstellen in austauschbare Objekte. Die Modell-Komponente ist für die Verarbeitung und Speicherung der Daten zuständig. Die View-Komponente stellt die Daten dar, in der Regel auf einem Bildschirm. Es können mehrere Views gleichzeitig existieren und auch verschiedene Visualisierungen derselben Daten zeigen, z. B. eine Tabelle und ein Balkendiagramm. Die Controller-Komponente ist schließlich für die Eingaben des Benutzers zuständig.

Abbildung 5.4 zeigt die Komponenten des MVC Konzepts. Der Benutzer tätigt Eingaben über den Controller, der die Eingabe auswertet und mit ihr das Modell ändert. Änderungen am Modell werden einem oder mehreren Views mitgeteilt. Der Benutzer erhält die Ausgaben durch die Views.

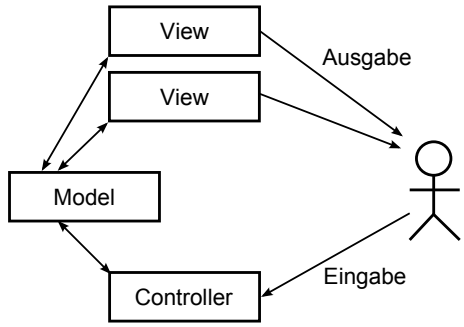


Abbildung 5.4: Komponenten des MVC Konzepts. Zwei parallele Views.

Presentation Abstraction Control (PAC) wurde 1987 von Coutaz et al. vorgestellt [39, 23]. Die Komponenten sind ähnlich aufgebaut wie beim MVC Konzept. Das

Modell wird Abstraction genannt und Controller und View sind zu einer Komponente Presentation zusammengefasst. Eine neue Komponente, Control, steuert das Zusammenspiel von Presentation und Abstraction. PAC ermöglicht hierarchische Strukturen, indem die Control-Komponenten mehrerer PAC-Agenten miteinander verknüpft werden. Agenten von höheren Schichten müssen nicht unbedingt eine Abstraction oder eine Presentation besitzen; reine Control-Agenten sind ebenfalls möglich. Der hierarchische Aufbau lässt sich beispielsweise bei Fenstersystemen nutzen. Ein PAC-Agent modelliert den Inhalt eines Fensters, ein übergeordneter Agent die Bildlaufleisten. Ein weiterer Agent, der nur aus einer Control-Komponente besteht, steuert das gesamte Fenster. Abbildung 5.5 illustriert dieses Beispiel.

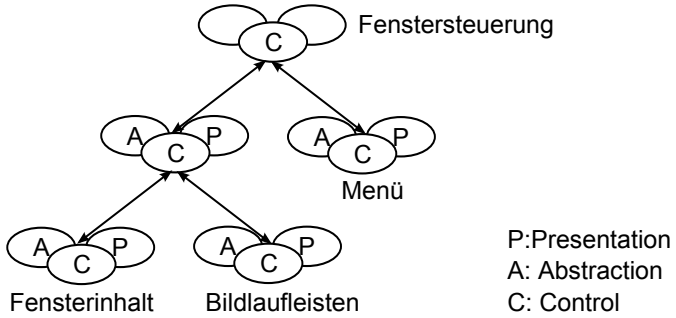


Abbildung 5.5: PAC-Agenten in hierarchischer Anordnung. Die Agenten beschreiben die Teile eines Fensters. Der oberste Agent steuert das Fenster und besitzt weder Abstraction noch Presentation.

5.1.5 Kombination der Ansätze zum multimodalen Arch (mArch-Modell)

Werden MVC oder PAC allein eingesetzt, so erfolgt die Steuerung der Interaktion noch nicht durch ein Dialogmodell wie es das Arch-Slinky Konzept ermöglicht. Die Modelle von MVC und PAC dienen rein der Abstraktion von Anwendungsdaten, die dem Benutzer kommuniziert werden sollen. Ihr Vorteil ist, dass sie durch mehrere Views und Controller beziehungsweise durch mehrere Presentation-Komponenten die Benutzung mehrerer Modalitäten parallel ermöglichen. Durch die Kombination mit einem Arch-Slinky basierten UIMS lässt sich die ursprüngliche Idee einer Trennung zwischen Anwendung, Darstellung und Dialogsteuerung der User Interface Management Systeme weiterverfolgen. Die Schwächen der einzelnen Ansätze lassen sich durch die Kombination beheben.

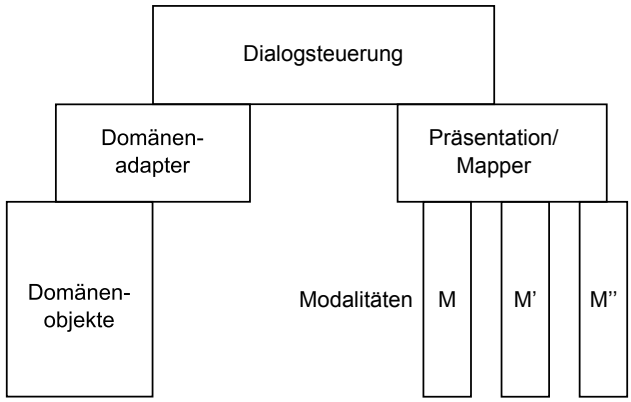


Abbildung 5.6: Die Kombination aus Arch-Slinky und multimodalen Präsentationskomponenten bildet das multimodale Arch-Modell (mArch).

In Abbildung 5.6 ist die kombinierte Architektur zu sehen. Das Konzept des Arch-Slinky Modells wird zur konzeptionellen Gliederung verwendet. Auf der rechten Seite verzweigt sich das Modell zu den einzelnen Modalitäten hin. Da mehrere Modalitäten parallel unterstützt werden und das Modell auf dem Arch-Slinky Modell basiert, wird es multimodales Arch-Modell (mArch-Modell) genannt.

Das mArch-Modell beinhaltet fast die gleichen Komponenten wie das Arch-Slinky Modell. Die Domänenobjekte werden im Domänenmodell durch Klassendiagramme beschrieben. Sie legen die Funktionalität der Anwendung fest.

Das Verhalten der Dialogsteuerung wird durch das Aufgaben- und das Dialogmodell beschrieben, die bereits in Kapitel 4 „Modellieren von Interaktionsanwendungen mit UML“ eingeführt wurden. Für das Aufgabenmodell werden Aktivitätsdiagramme, für das Dialogmodell Zustandsdiagramme eingesetzt.

Da die Verknüpfungen zwischen Domänenmodell sowie Aufgaben- und Dialogmodell bekannt sind, kann die Domänenadapter-Komponente kompakt realisiert werden. Sie leitet lediglich die Aufrufe an Domänenobjekte weiter.

Die Anbindung verschiedener Modalitäten geschieht über die Präsentationskomponente. Sie vermittelt zwischen dem abstrakten Dialogmodell und den verschiedenen Interaktionsobjekten der Modalitäten. Dazu verteilt sie die Ausgaben an alle Modalitäten und konzentriert die Eingaben in einen einzigen Ereignisstrom. Da die Präsentationskomponente ein 1:n Mapping von der Dialogsteuerung zu den Modalitäten vornimmt, wird sie in dieser Arbeit Mapper genannt. Der Mapper enthält Funktionalität, mit der plattformübergreifend zwischen den Kom-

ponenten kommuniziert werden kannn. Diese Funktionalität ist notwendig, da Modalitäten auf unterschiedlichen Plattformen zum Einsatz kommen.

Die einzelnen Modalitäten lassen sich als hierarchische PAC-Agenten oder mit dem MVC-Konzept als Kombination von View und Controller realisieren. In beiden Fällen liegen Abstraction bzw. Modell bei der Dialogsteuerung. In dieser Arbeit wird der MVC-Ansatz verwendet, um Modalitäten als Komponenten zu erstellen. Bei einer Generierung der Benutzerschnittstelle zur Laufzeit kann der PAC-Ansatz eingesetzt werden. Er hat den Vorteil, dass eine mehrstufige Aufgliederung in Anteile einer Benutzerschnittstelle eingesetzt werden kann. Bei einer solchen Aufgliederung lassen sich analog zum in Abschnitt 3.2.3 vorgestellten Cameleon-Framework ähnliche Anteile über Modalitäten hinweg zusammenfassen. Ein PAC-Agent beschreibt beispielsweise eine Interaktion in Software auf der AUI-Ebene. Ein Unteragent beschreibt eine GUI. Unteragenten steuern das Anwendungsfenster und die verschiedenen Elemente, aus denen die GUI sich zusammensetzt. Eine solche Strukturierung ergibt sich durch die Gruppierungen in den Ausgangsmodellen und lässt sich bei einer automatischen Generierung nutzen.

5.2 Zusammenspiel der Komponenten

Nachdem im letzten Abschnitt die Komponenten des mArch-Modells beschrieben wurden, wird im Folgenden erläutert, wie die Komponenten zusammenarbeiten und welche Aufgaben in ihnen ausgeführt werden.

Die Komponenten können plattformübergreifend durch Kommunikationsmechanismen verbunden werden, damit Anwendungen mehrere Geräte und Plattformen für ihre Benutzerschnittstelle verwenden können.

5.2.1 Domänenobjekte und Domänenadapter

Die Domänenobjekte beschreiben die Anwendung und werden durch Klassendiagramme modelliert. Sobald das System gestartet wird, werden auch die Domänenobjekte initialisiert. Zur Laufzeit interagiert der Benutzer durch seine Eingaben mit den Aktivitäts- und Zustandsdiagrammen der Dialogsteuerung. An Transitionen zwischen Zuständen und in Aktionen sind die Methoden der Domänenobjekte referenziert. Durch das Aufrufen der Methoden ändert sich der interne Zustand der Anwendung.

Domänenobjekte können beliebig geartet implementiert werden, solange eine Anbindung zur Laufzeit sicher gestellt ist. Der Domänenadapter sorgt für diese Anbindung der Anwendung an die Dialogsteuerung und setzt dazu die notwendige Technik wie z. B. verteilte Objekte oder CORBA ein. Der Adapter kann auch die Sprachunabhängigkeit der Implementierung der Domänenobjekte sicherstellen, indem er einen entsprechenden Anbindungsmechanismus bereithält.

5.2.2 Dialogsteuerung

Die Dialogsteuerung verwendet Aktivitäts- und Zustandsdiagramme, um den Ablauf der Interaktion zu regeln. Die Diagramme müssen dazu beim Start einer Anwendung geladen und in ein geeignetes internes Format überführt werden.

Der Ablauf eines Dialogs erfolgt wie in Kapitel 4 „Modellieren von Interaktionsanwendungen mit UML“ beschrieben. Ausgehend von einem Aktivitätsdiagramm werden Aktionen gestartet. Sobald der Benutzer eine Aktion ausführt, wird die Benutzerinteraktion an die Dialogsteuerung gemeldet, die den internen Zustand des Dialogs aktualisiert. Die Dialogsteuerung wählt auf Basis des Interaktionsmodells die nächste Aktion aus und benachrichtigt die Präsentationskomponente. Sie überträgt den eindeutigen Namen der Aktion, welche von der Präsentationskomponente an alle Modalitäten weiter vermittelt wird. Die Komponenten für die einzelnen Modalitäten nehmen daraufhin eine Aktualisierung ihrer Benutzerschnittstellen vor.

5.2.3 Präsentationskomponente/Mapper

Die Präsentationskomponente vermittelt zwischen den Modalitäten und der Dialogsteuerung. Wie bereits in Abschnitt 4.5.3 ausgeführt, lassen sich die Namen[1] von Aktionen und Zuständen für ein Mapping auf Teildialoge nutzen. Die Dialogsteuerung sendet daher den Namen der aktuellen Aktion oder des aktuellen Zustands an den Mapper und dieser verteilt ihn an alle beteiligten Modalitäten.

In der Gegenrichtung kann jede Modalität Eingaben des Benutzers an die Präsentationskomponente senden. Dieser muss die Eingaben des Benutzers verarbeiten, damit z. B. widersprüchliche Operationen abgefangen werden können.

Die Kommunikation zwischen Mapper und Modalitäten geschieht durch den Austausch von Nachrichten. Jede Modalität sendet Nachrichten an den Mapper, sobald ein Benutzer eine Interaktion ausführt. Im Gegenzug sendet der Mapper Nachrichten an die Modalitäten, sobald Aktionen oder Zustände betreten werden. Das Senden von Nachrichten entspricht einem Event-Mechanismus, ist aber nicht an ein Betriebssystem oder eine Plattform gebunden. Durch den universelleren Mechanismus ist es einfacher, verschiedene Hardware- und Software-Plattformen für Modalitäten zu kombinieren.

5.2.4 Modalitäten

An den Mapper können mehrere Modalitätskomponenten angebunden werden. Unter Umständen kann sogar eine Modalität mehrfach angeboten werden. Dies

[1] Die Namen sind eindeutig und erfüllen den Zweck von IDs.

ist der Fall, wenn ein Benutzer z. B. mehrere Geräte mit Display nutzen will, auf denen die gleiche Aufgabe erledigt werden soll.

In Abschnitt 3.2.3 wurden im Cameleon Framework verschiedene Ausprägungen für eine Benutzerschnittstelle vorgestellt. Es ist möglich, dass eine Benutzerschnittstelle als Concrete User Interface oder als Final User Interface vorliegt. CUIs verwenden in der Regel Toolkits und beschreiben die Interaktionselemente einer Modalität durch Toolkit-Elemente. FUIs können Benutzerschnittstellen sein, die in Binärcode vorliegen oder für die keine Toolkit-Elemente zur Verfügung stehen. Beide Ausprägungen sind im Präsentationsmodell des Klassifikationsframeworks, dass in Abschnitt 3.3 vorgestellt wurde, enthalten. In beiden Fällen muss die Benutzereingabe in Nachrichten für den Mapper umgesetzt werden. Im Fall von Toolkits existieren in der Regel Event-Mechanismen, die verwendet werden können. Bei FUIs muss die verwendete Komponente entsprechende Event-Mechanismen enthalten, um mit dem Mapper kommunizieren zu können.

In früheren Arbeiten wurde eine direkte Verknüpfung von einzelnen Modalitäten untereinander versucht. Die Arbeit von Mynatt und Edwards aus dem Jahr 1992 beschreibt beispielsweise die Abbildung von GUIs in Sprachschnittstellen durch eine starre Unterordnung von Sprache unter GUIs [121]. Erst mit dem Cameleon Framework wird ab dem Jahr 2003 das Konzept mehrerer parallel aktiver Modalitäten in der Literatur populär. Der Vorteil einer Mapper-Komponente, wie sie in dieser Arbeit Einsatz findet, liegt darin, dass jede Modalität für sich beschrieben werden kann, ohne dass Rücksicht auf Verknüpfungen zu anderen Modalitäten genommen werden muss.

Die einfachste Umsetzung einer Modalität lässt sich durch den Einsatz von *Templates* für Aktionen und Zustände realisieren. Die Nachrichten der Dialogsteuerung über aktuelle Aktionen und Zustände werden mit den Templates verknüpft und von der Modalitätskomponente zu den passenden Ereignissen ausgegeben. In Kapitel 6 „Realisierung“ wird dieses Verfahren an Hand eines Beispiels demonstriert.

5.3 Fission und Fusion

Die Mapper-Komponente verknüpft mehrere Modalitäten mit der Dialogsteuerung. Fusion bedeutet, dass Eingaben eines Benutzers miteinander verschmolzen werden. Eine Fusion ist bei der Verwendung von mehreren Modalitäten notwendig, da diese durch ungenügende Erkennung der Benutzereingabe widersprüchliche Eingaben liefern können. Unter Fission versteht man die Aufspaltung der Ausgabe. Eine Fission geschieht immer, wenn sich der interne Zustand der Dialogsteuerung ändert und die Zustandsänderung an die einzelnen Modalitäten kommuniziert werden muss. Die vorgestellte Mapper-Komponente übernimmt die Verteilung der Informationen. In der Literatur wird der Begriff weiter gefasst, es werden auch Verfahren darunter verstanden, bei denen die Benutzerschnittstellen der einzelnen Modalitäten zur Laufzeit generiert werden.

Im Folgenden wird beschrieben, wie die Fission hin zu einzelnen Modalitäten geschehen kann und wie die Generierung von Benutzerschnittstellen durch die verschiedenen Modelle beeinflusst wird. Im anschließenden Abschnitt wird auf die Fusion der Eingabe eingegangen. Weil diese Arbeit den Fokus auf die Beschreibung des Verhaltens von modellbasierten Benutzerschnittstellen legt, wird lediglich ein kompakter Überblick über bestehende Arbeiten zu den Themen Fusion und Fission gegeben.

5.3.1 Fission

Das Konzept der Fission ist die Verteilung von Ausgaben an alle beteiligten Modalitäten. Dazu werden die Nachrichten der Dialogsteuerung von der Mapper-Komponente kopiert und an alle verfügbaren Modalitäten gesendet.

Es gibt verschiedene Möglichkeiten, wie die Nachrichten verarbeitet werden können. Die einfachste Möglichkeit ist der Einsatz von fertigen *Templates*, die für die verschiedenen Dialogschritte ausgegeben werden.

Alternativ kann die gesamte Benutzerschnittstelle entweder vor der Benutzung des Systems oder fortlaufend zur Laufzeit generiert werden. Der erste Ansatz ist mit der *Model Driven Architecture* (MDA) verwandt [135]. Dabei entspricht das Platform Independent Model (PIM) der Interaktionsbeschreibung und das Platform Specific Model (PSM) einem erzeugten System mit eingebundenen Benutzerschnittstellen für bestimmte, im Voraus festgelegte Modalitäten. Der Ansatz, die Benutzerschnittstelle vor der Benutzung zu erzeugen, hat den Vorteil, auch auf wenig leistungsfähigen Endgeräten eingesetzt werden zu können. Nachteilig an diesem Ansatz ist unter anderem, dass zur Laufzeit keine weiteren Modalitäten eingebunden werden können und eine Anpassungen an nicht vorhergesehene Endgeräte nicht möglich ist. Der Ansatz der fortlaufenden Generierung führt die Transformation auf ein PSM kontinuierlich zur Laufzeit durch. Vorteilhaft ist, dass bei der fortlaufenden Generierung auf Veränderungen im Kontextmodell eingegangen werden kann. Ein weiterer Vorteil ist, dass Modalitäten während der Laufzeit eingebunden werden können. Von Nachteil ist die größere benötigte Rechenleistung, um die Transformationen durchführen zu können.

In dieser Arbeit werden Templates eingesetzt, da sie die Konzepte der Arbeit demonstrieren. Die entwickelten Modelle und Verfahren sind jedoch auch für beide generative Ansätze einsetzbar.

Kontextmodelle – wie sie in Abschnitt 3.1.1 vorgestellt wurden – haben Einfluß auf die Transformationen. An erster Stelle steht dabei das Gerätemodell, welches Informationen über die verwendbaren Ausgabemöglichkeiten und Interaktionselemente eines Endgeräts liefert. Das Benutzermodell beeinflusst durch Aussagen über den Benutzer die Ausgaben durch eine Modalität. Teile des Benutzermodells und des Umgebungsmodells zielen auf einen Einsatz zur Laufzeit ab. Anpassungen an eine besonders laute oder gleißend helle Umgebung sollten beispielsweise direkt während der Benutzung geschehen.

Verschiedene Arbeiten befassen sich mit der Generierung von Benutzerschnittstellen. Nylanders Arbeiten aus den Jahren 2003 und 2005 beschreiben beispielsweise, wie aus einer Beschreibungssprache zur Spezifikation von Webservices GUIs erstellt werden können [133, 134]. Eine andere Arbeit, die hier exemplarisch herausgegriffen wird, ist von van den Bergh et al., die eine eigene Beschreibungssprache für Benutzerschnittstellen beschreiben und für Mehrbenutzer-Systeme einsetzen [13]. Die Arbeit verwendet ConcurTaskTrees zur Beschreibung von Benutzerinteraktion und abstrakte Templates, aus denen die Benutzerschnittstellen zur Laufzeit erzeugt werden.

Für die Generierung von Benutzerschnittstellen sind schließlich automatische Layoutverfahren für GUIs wie im Überblick von Lok et al. beschrieben [100] oder die bereits vorgestellten Beautifications von Pederiva et al. aus Abschnitt 4.5.2 [151] wichtig.

5.3.2 Fusion

Die andere wichtige Aufgabe der Mapper-Komponente ist die Fusion von Eingaben. Eine Fusion der Eingabe ist wichtig, da ein Benutzer z. B. fast zeitgleich dieselbe Eingabe durch eine GUI und Sprache tätigen kann. Im Beispiel des MP3-Players kann der Benutzer eine Pause auslösen wollen, aber durch das zweite Kommando läuft die Musik weiter. Eine Behandlung der Eingaben ist also notwendig.

Die Fusion vereinigt die verschiedenen Eingaben der Modalitäten zu einem einzigen Eingabestrom. Im einfachsten Fall geschehen von einander unabhängige Eingaben, die durch den Mapper zur Dialogsteuerung weitergereicht werden. Der kompliziertere Fall ist, dass mehrere Eingaben vorhanden sind, die kombiniert werden müssen. Es kann sich in diesem Fall um mehrmals die gleiche Information oder um widersprüchliche Informationen handeln.

Eine ausführliche Behandlung von Fusionsmechanismen findet sich z. B. in der Dissertation von Nigay [125] aus dem Jahr 1994. Sie unterscheidet grundsätzlich zwei Möglichkeiten der Eingabe. Die Eingabe kann sich zeitlich überlappen oder es können zwei Eingaben sehr dicht aufeinander folgen. Wie von Nigay und Coutaz beschrieben, wird in beiden Fällen betrachtet, ob die Eingaben sich auf das gleiche Interaktionselement beziehen. Ist dies nicht der Fall, so sind die Eingaben unabhängig von einander. Andernfalls wird anschließend untersucht, ob eine Eingabe mehrfach vorgenommen wurde oder ob ein Widerspruch zwischen den Eingaben besteht [127]. Auch in der Arbeit von Tourraine et al. wird eine Fusion nach zeitlicher Nähe vorgenommen [189]. Die Arbeit nimmt dabei Rücksicht auf den Typ der Eingabe, um z. B. Sprachbefehle mit Gesten zu einer einzelnen Eingabe kombinieren zu können und so transmodale Eingaben zu ermöglichen. Die Arbeit von Cashera et al. aus dem Jahr 2008 betrachtet zusätzlich die Sicherheit, mit der Eingaben erkannt worden sind [33]. Sprach- oder Gestikeingaben werden vom jeweiligen Erkenner mit einer Bewertung der Wahrscheinlichkeit

geliefert. Die Eingaben durch verschiedene Modalitäten werden während der Fusion unter Berücksichtigung ihrer Wahrscheinlichkeiten verknüpft und dann die wahrscheinlichste Eingabe als Ergebnis verwendet.

5.4 Fazit

Die ersten Kapitel der Arbeit analysierten den Problemraum, führten ein Klassifikationsframework ein und wählten UML als Notation für abstrakte Interaktionsbeschreibungen. Im vorangegangen Kapitel 4 „Modellieren von Interaktionsanwendungen mit UML“ folgte eine detaillierte Verfahrensbeschreibung, wie mit UML Interaktionsanwendungen modelliert werden können.

Dieses Kapitel analysiert bisherige Ansätze von Architekturen für multimodale Anwendungen. Die bisherigen Ansätze sind konzeptioneller Art und nur bedingt für multimodale Anwendungen, die zugleich verteilte Modalitäten einsetzen können, geeignet. Mit dem mArch-Modell wurde eine Synthese aus dem Arch-Slinky Ansatz und multimodalen Präsentationskomponenten vorgestellt. Diese Software-Architektur ist für multimodale Anwendungen mit verteilten Modalitäten geeignet und ermöglicht den plattformübergreifenden Einsatz von Anwendungen.

Verfahren zur Fusion und Fission werden in dieser Arbeit ausgeklammert. Sowohl Fusion als auch Fission sind umfassende Themenfelder, welche deutlich über das Gebiet dieser Arbeit hinausgehen. Die Funktionsweisen beider Verfahren wurden beschrieben und es wurde verdeutlicht, an welchen Stellen Anknüpfungspunkte zur vorliegenden Arbeit existieren.

Zum Abschluss der Arbeit stehen noch eine Umsetzung des mArch-Modells in ein eigenes Software-Framework und eine Illustration anhand der Fallbeispiele aus.

6 Realisierung eines Software-Frameworks

> Eine Idee, die entwickelt und umgesetzt wird ist wichtiger als eine Idee, die nur als Gedanke existiert.
>
> *Buddha*

In den vorangegangenen Kapiteln wurde erarbeitet, wie mit Modellen Benutzerschnittstellen beschrieben werden können, wie dazu UML einsetzbar ist und welche Architektur benötigt wird. In diesem Kapitel wird eine Realisierung des mArch-Modells beschrieben. Um die Umsetzung von Anwendungen auf Basis von UML-Modellen zu vereinfachen, wurde ein eigenes Software-Framework, im Folgenden mArch-Framework genannt, erstellt.

Zu Beginn des Kapitels wird ein Überblick über die Komponenten des mArch-Frameworks gegeben. Anschließend wird beschrieben, wie sich das Framework beim Start einer Anwendung verhält. Während der Startphase werden die UML-Diagramme eingelesen und alle Komponenten des mArch-Frameworks initialisiert.

Nach der Beschreibung der Startphase wird das Verhalten zur Laufzeit erläutert. Zuerst wird auf die Ausführung von Aktivitätsdiagrammen eingegangen und der verwendete Algorithmus vorgestellt. Danach folgt der Algorithmus für die Ausführung von Zustandsdiagrammen. Die Anbindung von Klassendiagrammen dient zur Bereitstellung von Funktionalität. Schließlich wird auf das Präsentationsmodell und damit verbunden auf Fusion und Fission eingegangen.

Zum Ende des Kapitels wird noch ein Überblick über die Verwendung des mArch-Frameworks gegeben. Zusätzlich wird beschrieben wie neue Modalitäten eingebunden werden können.

6.1 Aufbau des User Interface Management System-Frameworks

Das Software-Framework für das mArch-Modell wird durch eine Reihe von Klassen realisiert. Abbildung 6.1 zeigt eine Übersicht über die wichtigsten Klassen, durch welche die Umsetzung erfolgt.

Die Klassen `Application`, `UIBuilder`, `Mapper` und `CUI` im unteren Bereich der Abbildung bilden das mArch-Modell nach.

Das Domänenmodell ist im linken Teil der Abbildung lokalisiert. Die jeweilige Anwendung, z. B. ein Getränkeautomat oder ein MP3-Player, wird durch die Klasse `Application` repräsentiert. Ihre Oberklasse `ThreadedApp` kapselt die Anwendung in einen eigenen Thread, so dass sie parallel zur Dialogsteuerung und den anderen Komponenten ausgeführt werden kann. Eine Klasse für die Domänenadapter-Komponente ist nicht notwendig, da alle Anwendungsaufrufe direkt durch Methoden von Domänenobjekten der Klasse `Application` bereitgestellt werden.

Die Dialogsteuerung steht im Mittelpunkt des Diagramms. Sie wird durch die Klasse `UIModel` implementiert und leitet sich von `XMI` ab, einer Oberklasse, die die UML Diagramme parst. Die Klasse `XMI` benötigt eine Reihe von Hilfsklassen, um beim Parsen eines XMI-Diagramms die entsprechenden UML-Entitäten erzeugen zu können. Im Paket UML sind die Hilfsklassen dargestellt. Es existieren Klassen für Zustände (`State`), die zugehörigen Übergänge (`Transition`) und auslösenden Ereignisse (`Trigger`), Aktionen von Aktivitätsdiagrammen (`Node`) und deren Kanten (`Edge`). Die Oberklasse `XMI` beachtet beim Parsen die hierarchische Strukturierung der Diagrammelemente. Dies ist notwendig, da Elemente eines Diagramms durch weitere Diagramme beschrieben werden können. Die Klasse `XMI` speichert Referenzen auf alle angelegten Objekte aus dem UML-Paket in sich, die von der Klasse `UIModel` für die Ausführung benötigt werden.

Die Klasse `Mapper` implementiert die Mapper-Komponente. Der Mapper verbindet die abstrakte Dialogbeschreibung mit den verschiedenen Benutzerschnittstellen.

Benutzerschnittstellen werden durch Modalitäten realisiert, die von der Klasse `CUI` (*Concrete User Interface*) abgeleitet werden. Es wurden drei verschiedene Modalitäten implementiert: eine grafische Schnittstelle `GladeUI`, eine Sprachschnittstelle `SpeechUI` sowie eine Schnittstelle `WiiUI` für tangible Interaktion durch Wii Remotes. Von jeder der abgeleiteten Klassen werden je nach vorhandenen Geräten Instanzen erzeugt. Diese verwenden eine modalitätsgebundene Beschreibung der Benutzerschnittstelle, damit ein Mensch mit ihr interagieren kann.

Die Klasse `UIBuilder` ist eine Hilfsklasse, die für eine Anwendung die Objekte für Dialogsteuerung, Mapper und alle verwendeten Benutzerschnittstellen anlegt. Für die Instanziierung der Benutzerschnittstellen wird das *Builder*-Pattern

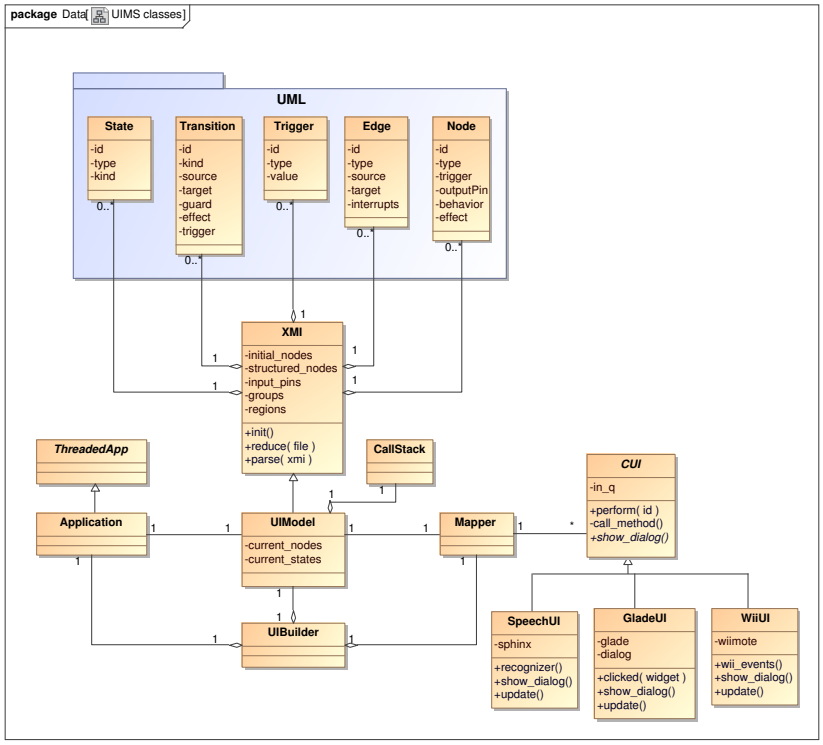


Abbildung 6.1: Umsetzung des mArch-Frameworks mit beteiligten Klassen.

verwendet. Entsprechend der vorhandenen Ein- und Ausgabegeräte werden Objekte für die einzelnen Modalitäten angelegt.

6.1.1 Parsen von UML Diagrammen

Zum Austausch von UML-Diagrammen wurde 2003 von der Object Management Group das *eXtensible Markup Language Metadata Interchange* (XMI) Format eingeführt [139]. Bei XMI handelt es sich um ein offenes, XML-basiertes Austauschformat, in dem ein Diagramm in seiner logischen Form und zusätzlich eine Beschreibung der Positionen aller Diagrammelemente für die grafische Repräsentation abgelegt wird.

UML Diagramme lassen sich zwar von Hand zeichnen, XMI-Dateien werden jedoch meist mit Werkzeugen erstellt. Es exisitiert eine sehr große Menge an Werkzeugen. Jeckle stellte bereits 2004 eine Liste von über 100 UML-Tools zur Bearbeitung von Diagrammen zusammen [84]. Für diese Arbeit wurde das Programm MagicDraw ausgewählt [129], da es alle Diagrammarten von UML auch in der Sprachversion 2.2 unterstützt.

Alle Elemente der UML werden durch die *Meta-Object Facility* (MOF) spezifiziert [141]. Die MOF beschreibt dabei jedes der Elemente durch eigene Klassen. Es besteht somit die Möglichkeit, nur die MOF-Elemente zu nutzen und einen sehr allgemeinen Parser zu erstellen, welcher sich an unterschiedliche Versionen der UML durch Auswerten des jeweiligen Meta-Modells anpassen kann. Da in den letzten Versionen lediglich Details im Meta-Modell verändert wurden, ist der höhere Aufwand für einen konfigurierbaren Parser nicht gerechtfertigt. Für diese Arbeit wurde darum ein Parser entwickelt, der an die aktuelle Version 2.2 der UML angepasst ist. Der Parser in der Klasse `XMI` erzeugt Instanzen der Klassen aus dem Paket UML, das in Abbildung 6.1 beschrieben wurde. Um ein modelliertes Diagramm ausführen zu können, genügen wenige Klassen sowie ein Teil ihrer Attribute. Attribute wie z. B. der Name eines Zustands müssen nicht geparst werden, da sie für die spätere Ausführung keine Rolle spielen. Durch die Beschränkung auf die relevanten Informationen können schlanke Klassen erstellt werden, mit denen effizient gearbeitet werden kann. Das Parsen der UML-Diagramme wird im Folgenden ausführlicher beschrieben.

Die Klasse `XMI` parst die Diagramme und bereitet sie für die weitere Verwendung auf. UML-Diagramme wachsen in ihrer XML-Repräsentation schnell in der Größe. Deshalb werden sie meist komprimiert gespeichert. Die Klasse `XMI` erkennt am Dateityp die Komprimierung und führt gegebenenfalls eine Dekompression durch [181]. Die XML-Repräsentation enthält in ihrer Struktur noch eine Reihe weiterer Elemente, die nicht benötigt werden. Vor dem eigentlichen Parsen wird deshalb zuerst die Datei verkleinert, indem alle Elemente, die für die zeichnerische Repräsentation zuständig sind, entfernt werden. Wie in Tabelle 6.1 zu sehen ist, verkleinert sich die XML-Repräsentation in der Regel um mehr als 90%.

	Dateigröße		Zeilen		
Diagramm	vorher	nachher	vorher	nachher	Reduktion
Getränkeautomat	516,4 kB	20,3 kB	9816	630	96,1 %
MP3-Player	547,5 kB	17,9 kB	10514	600	96,7 %

Tabelle 6.1: Einsparungen durch Reduktion der UML-Diagramme (Prozentsatz auf Dateigröße bezogen).

Es gibt zwei Möglichkeiten, die vorliegenden Diagramm zu parsen, Tree- oder Stream-basiert. Tree-basierte Parser lesen das komplette XML-Dokument ein und erzeugen daraus einen Baum im Hauptspeicher. Das W3C hat für diese Zugriffsart mit dem *Document Object Model* (DOM) eine Schnittstelle entwickelt

[213, 211]. Beim Stream-basierten Parsen werden nur Funktionen definiert, die aufgerufen werden, sobald der Parser auf ein bestimmtes Tag trifft. Auch hierfür existiert mit der *Simple API for XML* (SAX) eine standardisierte Schnittstelle, die durch eine Gruppe informell kooperierender Autoren der XML-DEV Mailingliste entstand [106]. SAX-Parser sind ressourcenschonender, da nicht das komplette Dokument in Strukturen im Speicher gewandelt werden muss. Dafür ist die Navigation im XML-Dokument erschwert. DOM-Parser haben einen höheren Ressourcenverbrauch, da das komplette Dokument eingelesen wird. Ihr Vorteil ist, dass sehr einfach auf alle Teile eines XML-Dokuments zugegriffen werden kann.

Durchläufe	Parser	total	real	kBps
2	null	0.470000	0.477970	29810.78
(14 MB)	nokogiri	1.400000	1.408751	10114.39
	libxml-ruby	1.710000	1.721740	8275.73
	rexml	151.040000	151.647898	93.96
100	null	21.880000	21.956262	32447.83
(700 MB)	nokogiri	89.060000	89.409650	7968.19
	libxml-ruby	98.750000	99.134173	7186.55

Tabelle 6.2: Vergleich der Performance verschiedener XML-Parser. Quelle: [148] (gekürzt).

Durch die Reduktion um die nicht benötigten Elemente lassen sich XMI-Dateien auf eine Größe reduzieren, die problemlos den Einsatz eines DOM-Parsers zuläßt. In der aktuellen Implementierung wird Ruby Electric XML (REXML) aus der Standardbibliothek von Ruby verwendet [167]. Bei großen Anwendungen wachsen die Diagramme in ihrer Komplexität. In diesem Fall lässt sich der Parser durch eine kompatible schnellere Variante ersetzen, die gesondert installiert werden muss. In Frage kommt z. B. Nokogiri, der derzeit schnellste XML-Parser für Ruby [149]. Wie man in Tabelle 6.2 sieht benötigt REXML über 2 Minuten um 14 MB XML zu parsen, Nokogiri hingegen nur 1,4 Sekunden. Ein so großes Diagramm müsste bereits über 700-mal mehr Elemente als der in Kapitel 5 vorgestellte Getränkeautomat enthalten. Bei großen Datenmengen wie beim Parsen von 700 MB benötigt Nokogiri knapp 1,5 Minuten. Die Zeit für das Parsen fällt lediglich einmal, beim Start einer Anwendung, an.

Aus einem XML-Dokument werden durch eine Reihe von *XPath*-Ausdrücken die Informationen extrahiert, die für die Erstellung der UML-Elemente aus den Diagrammen nötig sind. Die XML Path Language (XPath) [209] ist eine Abfragesprache, die vom W3-Konsortium entwickelt wurde. Für jedes unterstützte Modellelement der UML (State, Transition, Node, Edge und Trigger) existiert eine eigene Klasse im aus dem UML-Paket aus Abbildung 6.1, in welcher alle relevanten Informationen durch Attribute repräsentiert werden. Der Parser erzeugt für jedes Element aus dem Modell ein eigenes Objekt und speichert zusätzlich Infor-

mationen, die später für der Ausführung benötigt werden, z. B. welche Zustände Startzustände sind.

Für alle Diagramme aus einer XMI-Datei müssen alle Startzustände ermittelt werden. Dies ist notwendig, da z. B. ein hierarchischer Zustand mehrere Startzustände enthalten kann. Weiter ist es wichtig, zu wissen, welche Elemente die Startknoten des hierarchisch höchsten Diagramms sind. Es besteht die Möglichkeit, dass mehrere solcher Elemente existieren. In der Implementierung des Software-Frameworks ist vorläufig nur ein Knoten in einem Aktivitätsdiagramm vorgesehen. Als weitere Vereinfachung muss der Knoten den Namen „start“ besitzen.

6.1.2 Initialisierung des Frameworks

Bevor eine Anwendung ihre Benutzerschnittstelle durch eine oder mehrere Modalitäten bereitstellt, müssen alle Komponenten initialisiert werden. Die Klasse `UIBuilder` erzeugt alle benötigten Komponenten und verbindet sie miteinander.

Input : Referenz auf $anwendung$, Pfad zu XMI, Modalitäten
 # Lege Array zur Threadverwaltung an
1 threads ← Array.new()
 # Erzeugen von Dialogmodell $uims$ und Mapper
2 uims ← UIModel.new(XMI)
3 mapper ← Mapper.new()
 # Verbinden der Komponenten miteinander
4 mapper.connect_uims(uims)
5 uims.connect_app(anwendung)
6 uims.connect_mapper(mapper)
 # Registrieren der Modalitäten beim Mapper
7 **foreach** *mod* ∈ *modalities* **do**
8 threads.add(mod.thread)
9 mapper.register(mod)
10 **end**
 # Dialog starten
11 uims.start

Algorithmus 1 : Initialisierung des Frameworks durch `UIBuilder`.

In Algorithmus 1 ist der Ablauf der Initialisierung dargestellt. Es werden Komponenten zur Dialogsteuerung und Verwaltung der Modalitäten angelegt. Die Anwendungskomponente wird mit der Dialogsteuerung, die Dialogsteuerung mit dem Mapper und der Mapper mit allen Modalitäten verbunden. Schließlich wird der Dialog gestartet.

Im Sequenzdiagramm der Abbildung 6.2 ist das Verhalten des Software-Frameworks beim Start einer Anwendung zu sehen. Die Anwendung startet als erstes den `UIBuilder`, dieser erzeugt die Dialogsteuerung und den Mapper (2, 3). Beim Mapper registrieren sich anschließend die verfügbaren Modalitäten (4, 5). Nachdem das Dialogmodell initialisiert wurde, wird es vom `UIBuilder` aus gestartet (6). Das Dialogmodell verfolgt vom vorgegebenen Startknoten aus das Diagramm (7) und propagiert den aktuellen Dialogschritt an den Mapper (8). Dieser verteilt die Information an die Komponenten der beteiligten Modalitäten (9, 10). Damit ist die Anwendung bereit, Eingaben durch den Benutzer zu empfangen.

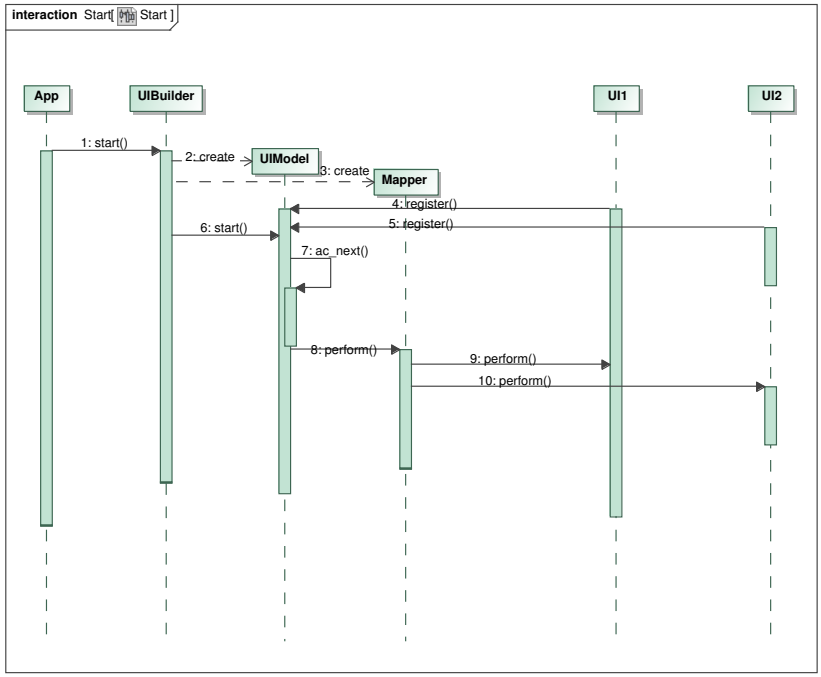


Abbildung 6.2: Sequenzdiagramm zum Verhalten des mArch-Frameworks beim Start einer Anwendung. UIs registrieren sich beim Builder, dieser initiiert anschließend den Interaktionszyklus.

6.2 Ausführen von UML

Es ist möglich, aus den geparsten UML-Diagrammen Code für eine Zielplattform zu kompilieren oder die Diagramme zur Laufzeit zu interpretieren, wie in Abschnitt 2.3 beschrieben wurde. Für das Kompilieren spricht die bessere Performance, dagegen spricht, dass Anpassungen in der Modellierung ein erneutes Kompilieren erzwingen. Für das Interpretieren spricht, dass Eingaben von Benutzern sich schnell genug verarbeiten lassen und Anpassungen in der Modellierung nur einen Neustart der Anwendung erfordern. Die Möglichkeit, zur Laufzeit verschiedene Benutzerschnittstellen durch Fissionsmechanismen zu erzeugen, ist ein wesentlicher Grund für das Interpretieren von Modellen. Daher werden in dieser Arbeit die Diagramme interpretiert. Das Interpretieren ist auch unter dem Namen *executable Models* bekannt. Die Modelle müssen dazu vorher auf Fehler überprüft werden, dies kann eine *Model checking*-Komponente vornehmen.

Für UML existieren bereits Arbeiten zu ausführbaren Modellen, z. B. von Mellor et al. [107]. Auch Zusammenhänge zwischen Generierung und ausführbaren Modellen wurden bereits untersucht [161]. Allerdings existieren noch keine vollständigen Interpreter für Aktivitäts- und Zustandsdiagramme. Eine Entwicklungsumgebung für Aktivitätsdiagramme, die während diese Arbeit entstand und um Zustandsdiagramme erweitert wurde, ist ActiveCharts. Die Entwicklungsumgebung ActiveCharts wurde von Sarstedt et al. vorgestellt und an der Universität Ulm entwickelt [170, 1]. Sie bietet über Aktivitätsdiagramme hinaus die Möglichkeit, Klassen und deren Methoden zu modellieren, die von Aktionen benutzt werden können. ActiveCharts verwendet bisher Microsoft Visio, um Diagramme zu erzeugen, und setzt Visio als Komponente in der Entwicklungsumgebung ein, um Diagramme auszuführen. Dadurch ist eine Bindung an eine Plattform vorhanden, in zukünftigen Versionen wird eine plattformunabhängige Umgebung angestrebt.

Abstract State Machines (ASMs) sind ein Ansatz, das Verhalten von UML-Diagrammen durch formale, verifizierbare Automaten zu beschreiben [21, 163]. Abstrakte Zustandsmaschinen beschreiben durch die Änderungen ihrer internen Zustände das Verhalten von einzelnen Diagrammen. Sowohl Aktivitäts- als auch Zustandsdiagramme können durch ASMs beschrieben werden, dadurch lassen sich die Einbettungen von Diagrammen in übergeordnete Diagramme ebenfalls nachbilden. Allerdings unterstützen ASMs noch nicht vollständig alle benötigten Diagrammelemente und deren Verhalten.

Zur Implementierung des Software-Frameworks wurde ein Interpreter für Aktivitäts- und Zustandsdiagramme entwickelt. Dieser zeigt die Realisierbarkeit des Konzepts. Im Folgenden wird die Funktionsweise des Interpreters vorgestellt.

In Abbildung 6.3 wird das Laufzeitverhalten des Frameworks gezeigt. Das UML-Diagramm ist in zwei alternative Abschnitte, mit *alt* bezeichnet, geteilt. Im oberen Abschnitt sind die Schritte beim Abarbeiten eines Übergangs in einem Aktivitäts- oder Zustandsdiagramm zu sehen. Der Interpreter verarbeitet den Übergang

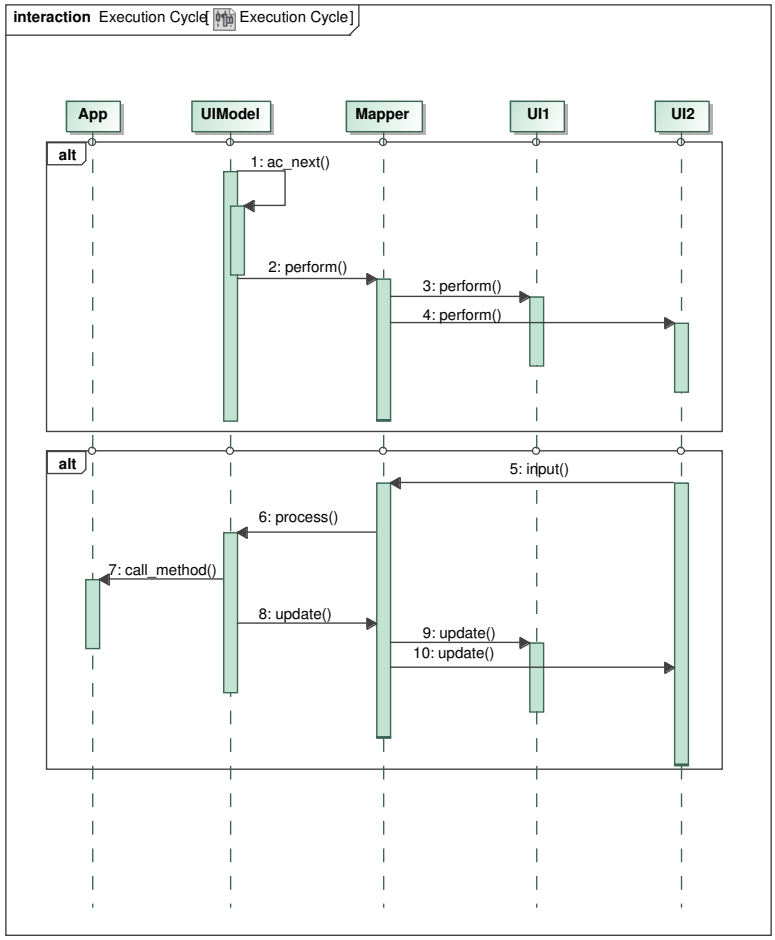


Abbildung 6.3: Verhalten des mArch-Frameworks zur Laufzeit. Die beiden Alternativen (*alt*) zeigen oben den Übergang zwischen zwei Aktionen und unten die Verarbeitung einer Benutzereingabe.

(1), wobei in diesem Fall kein Aufruf an das Anwendungsmodell erfolgt. Die Information über die aktuelle Aktion wird als Nachricht an den Mapper und über diesen an die Modalitäten übermittelt (2, 3, 4).

In der unteren Alternative tätigt der Benutzer eine Eingabe, die vom Framework und der Anwendung verarbeitet wird. Nach der Eingabe (5), wirkt diese sich auf das Modell (6) und die Anwendung (7) aus. Die interne Zustandsänderung der Dialogsteuerung wird durch Updates zu den Benutzerschnittstellen propagiert (8, 9, 10).

6.2.1 Hilfsmechanismus Kaktusstack

Um Aktivitäts- und Zustandsdiagramme auszuführen, wird eine besondere Art eines Stacks, ein sogenannter *Call Stack*, als Hilfsmechanismus eingesetzt. Ein Call Stack ist ein Stack, der Informationen über aktive Unterprogramme einer Anwendung speichert. Für Hochsprachen ist der Call Stack transparent, da Routinen für seine Verwaltung von Bibliotheken bereitgestellt werden. Die wichtigste Aufgabe eines Call Stacks ist, zu verwalten, an welcher Stelle nach einem Unterprogrammaufruf fortgefahren werden soll. Zum Beispiel kann ein Unterprogramm zur Berechnung der Quadratwurzel von mehreren Stellen in einem Programm aus aufgerufen werden. Der Call Stack sorgt dafür, dass nach einem solchen Aufruf das Programm an der Stelle danach fortgesetzt wird [4].

In dieser Arbeit wird der Call Stack zum Speichern der Position in den Diagrammen verwendet. In UML-Diagrammen können wegen Parallelität mehrere Aktionen oder Zustände gleichzeitig aktiv sein. Immer, wenn in einem Diagramm ein Element durch ein anderes Diagramm genauer spezifiziert wird, wird dies in einem Call Stack vermerkt. Da bei UML-Diagrammen durch Parallelität Verzweigungen entstehen wurde für diese Arbeit die Idee des Call Stacks um Verzweigungen erweitert. Diese Idee wurde bereits 1968 von Hauck und Dent beschrieben, die einen Stack mit zwei Enden in einer Implementierung in Hardware verwendeten [62]. Wegen der möglichen mehrfachen Verzweigungen beim Einsatz eines solchen Stacks für UML-Diagramme wird der Stack Kaktusstack genannt.

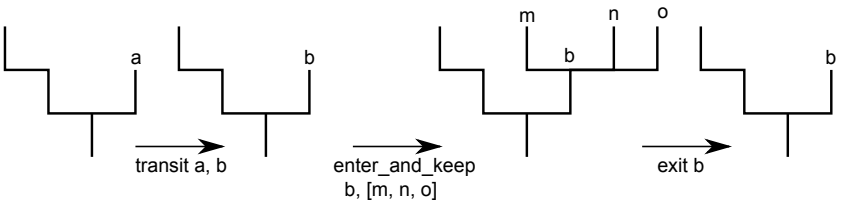


Abbildung 6.4: Kaktusstack mit möglichen Operationen.

In Abbildung 6.4 sind die verschiedenen Operationen zu sehen, die auf dem Kaktusstack möglich sind. Alle Kaktusstacks in der Abbildung besitzen mehrere Verzweigungen. Die einfachste Operation ist ein Übergang, d. h. in einem Diagramm wird von einem Zustand zu einem anderen (oder einer Aktion zur nächsten) gewechselt. Diese Operation wird `transit` genannt und ändert die Spitze einer Verzweigung von a nach b. Wird von einem Diagramm aus ein anderes Diagramm betreten, so bleibt die aufrufende Aktion bzw. der aufrufende Zustand aktiv. Dafür existiert die Operation `enter_and_keep`, die in der Abbildung den Knoten b des zweiten Stacks aktiv belässt und drei weitere Enden (m, n, o) zum Kaktusstack hinzufügt. Durch die Operation `exit` wird der Stack zurückgebaut. In der Abbildung bewirkt `exit` von Knoten b aus, dass die Knoten m, n und o vom Stack entfernt werden. Am Beispiel des MP3-Players entspricht diese Operation dem Verlassen der Aktion `sm-player`. Das enthaltene Zustandsdiagramm mit allen aktiven Regionen wird in diesem Fall abgebrochen.

6.2.2 Aktivitätsdiagramme

Nach der Einführung des Kaktusstacks wird im Folgenden beschrieben, wie Aktivitätsdiagramme ausgeführt werden. Die realisierte Dialogsteuerung verwendet nur einen Teil aller Modellelemente der Aktivitätsdiagramme: Activity, ActivityNode in verschiedenen Ausprägungen, ActivityEdge und ActivityGroup. Dieses Subset ist auf alle Elemente erweiterbar. Abbildung 6.5 zeigt einen Überblick über die wichtigsten verwendeten Elemente von Aktivitätsdiagrammen und die Relationen zwischen ihnen.

ActivityNodes können Aktionen (Actions) oder Kontrollknoten sein. Zur Steuerung von Abläufen lassen sich DecisionNodes, MergeNodes, ActivityFinalNodes und InitialNodes einsetzen. Die Verbindungen zwischen Aktionen und Steuerelementen werden durch die ActivityEdges beschrieben.

In der Implementierung des mArch-Frameworks werden alle Unterklassen von ActivityNodes zu Objekten der eigenen Klasse Node zusammengefasst. Da UML-Elemente nicht in ihrer vollen Komplexität unterstützt werden müssen, sondern lediglich wenige Attribute wie ID, Verhalten und Trigger für die Interpretation benötigt werden, ist es zweckmäßig, eine Vereinfachung vorzunehmen und dafür das Attribut `type` einzuführen. Durch dieses Attribut kann die Art eines Nodes unterschieden werden. Die Klassen der Realisierung wurden bereits in Abbildung 6.1 vorgestellt.

Eine Besonderheit stellen *Input Pins* dar, da sie zur Darstellung von Objektflüssen zwischen Actions benötigt werden. Alle bisherigen Verbindungen zwischen Elementen von Aktivitätsdiagrammen steuerten den Kontrollfluss. Bei einem Objektfluss zwischen zwei Actions startet dieser am Output Pin einer Action und endet am Input Pin der referenzierten Action. Input Pins beinhalten indirekt einen Kontrollfluss.

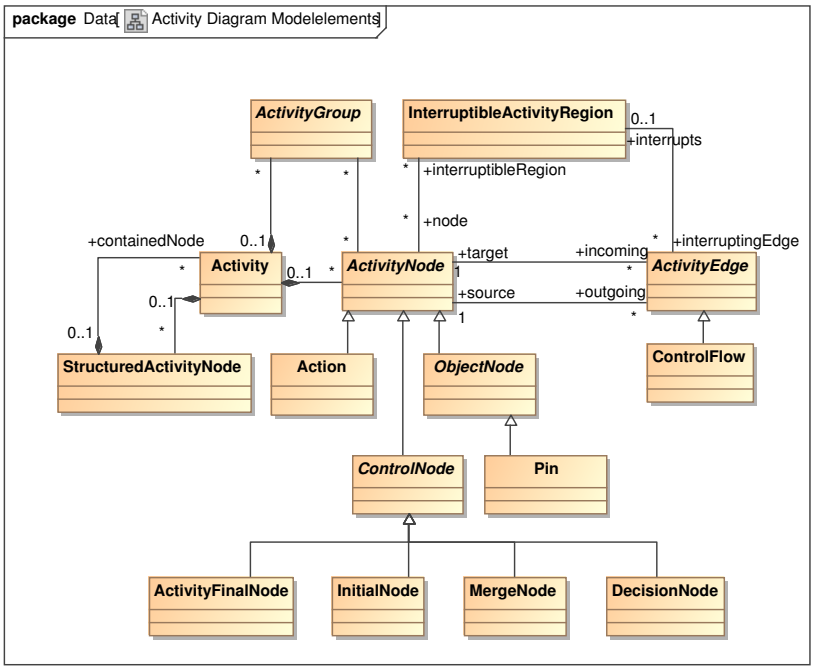


Abbildung 6.5: Die eingesetzten Modellelemente für Aktivitätsdiagramme nach [142].

Actions können komplex aufgebaut sein und andere Elemente enthalten. In diesem Fall spricht man von einer StructuredActivityNode. In der Realisierung wird gespeichert, welche anderen Elemente eines Diagramms zu einer StructuredActivityNode gehören.

Außer den Elementen für Aktionen müssen auch alle zugehörigen Übergänge gespeichert werden. Für jeden Übergang wird ein Objekt der Klasse `Edge` angelegt, welches die ID der Quelle und des Ziels speichert. Dies ist notwendig, da auf eine Edge nur von ihrem Quell-Node aus zugegriffen werden kann.

Die InterruptibleActivityRegions (unterbrechbarer Aktivitätsbereich) werden ähnlich wie StructuredActivityNodes mit Hilfe von Gruppen modelliert. Jede Region beinhaltet Aktivitätsknoten und -kanten und kann durch eine Edge mit dem Attribut `interrupts` unterbrochen werden, wobei die enthaltenen Nodes der Region terminiert werden.

```
   Input : current-node-id, stack
 1 current-edges ← edges[current-node-id]
 2 current-edges ← current-edges mit guard = true
   # Edges betrachten
 3 foreach edge ∈ current-edges do
 4     node ← nodes[edge.target]
 5     if edge.interrupts then stack.exit(edge.source, edge.target)
 6     if node.type ∈ CallBehaviorAction then
 7         stack.transit(edge.source, edge.target)
 8         stack.enter_and_keep(edge.target, node.behavior)
 9         start_diagram(node.behavior)
10     end
11     else if node.type ∈ CallOperationAction then
12         app.call(node.effect) stack.transit(edge.source, edge.target)
13     end
14     else if node.type ∈ ActivityFinalNode then
15         ac_id ← ID der zu beendenden Activity
16         stack.exit ac_id
17         if stack.empty then
               # Diagramm komplett verlassen
18             exit
19         else
               # Kontrolle an übergeordnetes Diagramm geben
20             ac_next stack.pop
21         end
22     end
23     else
           # Element noch nicht unterstützt
24     end
25     mapper.perform(node.name)
26 end
```

Algorithmus 2 : Algorithmus des Interpreters für Aktivitätsdiagramme.

Algorithmus 2 zeigt, wie Aktivitätsdiagramme verarbeitet werden. Der Algorithmus beginnt, sobald von einer Aktion aus ein Übergang stattfinden soll. Es werden nur Kanten betrachtet, die auch traversiert werden dürfen (2, 3). Es findet eine Fallunterscheidung statt. Bei einer Unterbrechung wird die aktuelle Gruppe von Aktionen abgebrochen (5). Falls ein weiteres Diagramm enthalten ist wird dieses aktiviert (6–10). Es kann jeweils nur eine neue Aktion betreten werden (11–13). Falls das aktive Diagramm beendet wird, kann entweder die Kontrolle an das aufrufende Diagramm übergeben werden, oder, falls es keines gibt, die Anwendung beendet werden (14–22). In Zeile 23 ist der Fall beschrieben, dass ein bisher noch nicht unterstütztes Element verwendet wird.

6.2.3 Zustandsdiagramme

Von Aktivitätsdiagrammen aus können, wie in Kapitel 4 bereits beschrieben wurde, Zustandsdiagramme aufgerufen werden. Diese beschreiben ausführlich das Zustandsverhalten in einer einzelnen Aktion.

Die Dialogsteuerung verwendet in ihrer aktuellen Umsetzung einen Teil aller möglichen Modellelemente von Zustandsdiagrammen. In Abbildung 6.6 sind sie im Überblick dargestellt.

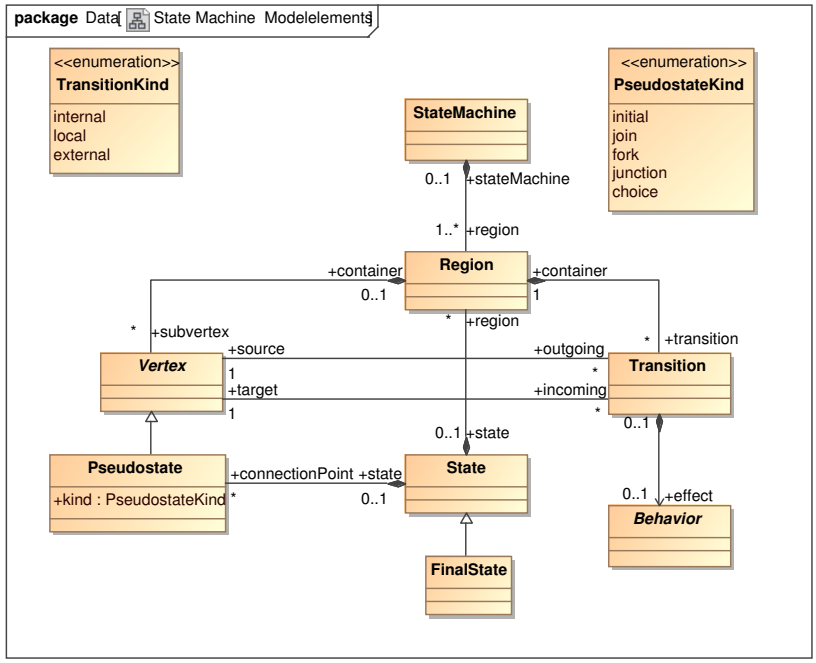


Abbildung 6.6: Benutzte Modellelemente für Zustandsmaschinen.

Ein Zustandsdiagramm enthält immer eine Zustandsmaschine. Die Zustandsmaschine enthält Zustände in einer oder mehreren Regionen, welche Zustände gruppieren. Regionen werden innerhalb von komplexen Zuständen auch für die Umsetzung von Nebenläufigkeit eingesetzt.

Eine Reihe von Elementen in Zustandsdiagrammen wird in UML durch *Pseudostates* (Pseudozustände) umgesetzt. Pseudostates sind Elemente wie Startknoten,

```
   Input : trigger, stack
   # Alle relevanten Transitionen sammeln
 1 transitions ← stack.top-elems mit ∀ transition.trigger=trigger
 2 foreach t ∈ transitions do
 3     if t.target.type = Pseudostate then transitions ∪ t.target
 4 end
 5 foreach trans ∈ transitions do
 6     if trans.trigger ≠ null and trigger="" then lösche trans
 7     if trigger ≠ trans.trigger then lösche trans
 8     if not app.test_guard(trans.guard) then lösche trans
 9 end
10 foreach trans ∈ transitions do
11     foreach effect ∈ trans.effect do
12         app.call(effect)
13         mapper.update(effect)
14     end
15     if trans.target ∈ state-regions then
16         stack.transit trans.source, trans.target
17         stack.enter_and_keep trans.target, state-regions[trans.target]
18         enter_region state-regions[trans.target]
19     else
           # Alle anderen Zustände
20         target ← states[trans.target]
21         if target.type = State then
22             stack.transit trans.source, trans.target
23             mapper.update target.name
24         end
25         if target.type = FinalState then
26             list ← Strang des Kaktusstacks mit Ende=trans.source
27             stack.exit list.pop
28             activity-diagram-algorithm(list.pop)
29         end
30     end
31 end
```

Algorithmus 3 : Algorithmus des Interpreters für Zustandsmaschinen.

Gabelung, Vereinigung und Entscheidung (*initial*, *fork*, *join*, *choice*). Ein Pseudostate besitzt einen Typ *kind*, der durch eine Aufzählung definiert wird. In Abbildung 6.6 sind die entsprechenden englischen Namen aus dem Standard festgehalten.

Übergänge zwischen Zuständen werden durch Objekte der Klasse Transition beschrieben. Transitionen verbinden Pseudostates, Regionen und Zustände untereinander und können ein Verhalten (*Behavior*) besitzen.

In der Realisierung werden Informationen über die Zugehörigkeit von Zuständen zu Regionen an separater Stelle gespeichert. In Abbildung 6.1 werden deshalb nur die Klassen State und Transition aufgeführt, um Zustandsdiagramme zu beschreiben. Jeder Zustand besitzt mehrere Attribute: Guard, Effect und Trigger.

Wird ein Zustandsdiagramm betreten, so kommt Algorithmus 3 zum Einsatz. Es werden alle Zustandsübergänge zu den Spitzen des Kaktusstacks gesammelt, deren Trigger dem aktuellen Trigger entsprechen (1). Diese Transitionen werden bis zum nächsten Zustand verfolgt, da sie durch Pseudozustände eine Kette bilden können (2–4). Anschließend werden die Transitionen herausgefiltert, die nicht gleich dem Trigger sind (5–9).

Nach diesen Schritten sind lediglich noch die begehbaren Transitionen übrig. Diese können ein Verhalten (Effect) besitzen, welches das Ausführen von Operationen auf der Anwendung bewirkt. Ist dies der Fall, wird über den Mapper eine Aktualisierung der Ausgabe vorgenommen (11–14).

Es können Regionen betreten werden, die Unterzustände enthalten. In diesem Fall muss der Kaktusstack aktualisiert werden (15–18). Ansonsten verbleiben noch zwei Fälle: ein einfacher Zustand[1] oder ein Endzustand kann erreicht worden sein. Bei einem einfachen Zustand wird die Transition auf dem Kaktusstack nachvollzogen und anschließend eine Aktualisierung der Ausgabe durchgeführt (21–24). Falls ein Endzustand erreicht wurde, wird der Strang des Kaktusstacks benötigt, von dem die Transition ausging. Dessen Zustandsmaschine wird beendet (27). Nach der Beendigung des Zustandsdiagramms ist im darüber liegenden Aktivitätsdiagramm eine Aktion beendet und von ihr aus muss die passende Edge zur nächsten Aktion traversiert werden. Zu diesem Zweck erfolg ein Aufruf für Algorithmus 2 (28).

6.2.4 Anbindung von Klassendiagrammen

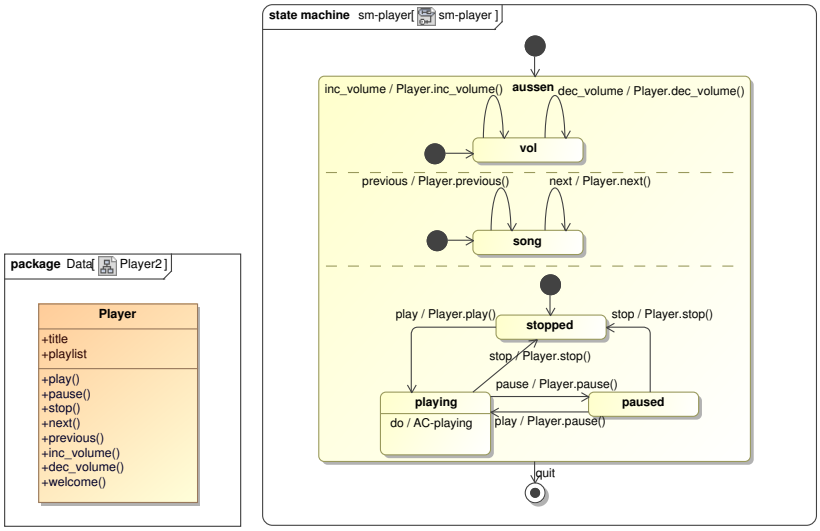
Sowohl in Zustands- als auch in Aktivitätsdiagrammen können Operationen auf Klassen integriert sein. Die Operationen werden bei einem Zustandsdiagramm durch *Effects* entlang von Transitionen oder innerhalb von Zuständen (*OnEnter*, *OnExit*, *do*) ausgeführt. In Aktivitätsdiagrammen können Aktionen als *CallOperationActions* auf Methoden von Klassen verweisen.

UML besitzt ein fertiges Mapping durch ein Binding von Behaviors an Methoden von Klassen. Zur Anbindung müssen daher nur die benötigten Methoden mit Transitionen oder Aktionen verknüpft werden.

Für die Modellierung von Anwendungen wird in der Regel eine Vielzahl von Klassen verwendet. Zur Beschreibung des Verhaltens der Benutzerschnittstelle einer Anwendung kann die Verwendung einer vollständigen Modellierung zu komplex sein. Für diesen Zweck lässt sich eine Klasse mit ausgewählten Methoden eines

[1] Ein Zustand, der keine Unterzustände enthält.

Subsystems erstellen. Sie delegiert die Funktionalität an andere Klassen des Subsystems und bietet durch die Auswahl an Methoden eine vereinfachte Schnittstelle. Eine solche Klasse implementiert das Fassaden-Pattern (engl. *facade-pattern*) von Gamma et al. [55]. Eine Klasse, die eine Fassade verwendet, kann nicht feststellen, dass diese andere Klassen verbirgt. Durch diesen Mechanismus lassen sich Klassen, die nicht für das Verhalten der Benutzerschnittstelle benötigt werden, verbergen. In Abbildung 6.7a ist die Klasse Player des MP3-Players zu sehen. Sie wirkt als Fassade und verbirgt die Implementierung der Playlist.



(a) Klassendiagramm MP3-Player

(b) Zustandsdiagramm MP3-Player

Abbildung 6.7: Links das Klassendiagramm des MP3-Players, rechts das Zustandsdiagramm.

In Abbildung 6.7b wird durch das Zustandsdiagramm das Verhalten der Benutzerschnittstelle des MP3-Players beschrieben. Entlang der Übergänge zwischen den verschiedenen Zuständen stehen die Trigger sowie die von ihnen bewirkten Operationen. Wenn beispielsweise der Trigger *play* den Übergang von *stopped* zu *playing* bewirkt, wird die Methode `play()` der Klasse Player aufgerufen.

6.2.5 Anbindung des Präsentationsmodells

Wie in Abschnitt 4.5.3 beschrieben, wird in der Realisierung das Präsentationsmodell durch ein händisches Mapping angebunden. Im selben Abschnitt wurde auch die Erstellung einer GUI mit dem GTK Toolkit ausgeführt. In diesem Abschnitt folgt die Beschreibung der Umsetzung der einzelnen Modalitätskomponenten des mArch-Modells.

Eine Benutzerschnittstelle wird von der Klasse `CUI` abgeleitet. Diese Klasse stellt Methoden zur Kommunikation mit der Klasse `Mapper` zur Verfügung. Da sich die einzelnen Modalitäten über mehrere Plattformen verteilen können, muss zwischen den Komponenten kommuniziert werden. Für diesen Zweck werden verteilte Objekte der distributed Ruby Bibliothek *druby* eingesetzt [174]. Von der Klasse `CUI` abgeleitete Klassen verbinden sich automatisch durch Aufruf des Konstruktors der Oberklasse mit dem Mapper. Sie besitzen zwei Queues für Nachrichten, eine zum Senden und eine zum Empfangen. Im Folgenden wird am MP3-Player exemplarisch beschrieben, wie solche Komponenten zusammenwirken. Im Prototyp sind drei Modalitäten implementiert, eine grafische Schnittstelle, eine Sprachschnittstelle sowie eine Schnittstelle für tangible Interaktion durch Wii Remotes. Jede dieser Komponenten wird durch eine eigene Klasse implementiert, wie Abbildung 6.8 zeigt.

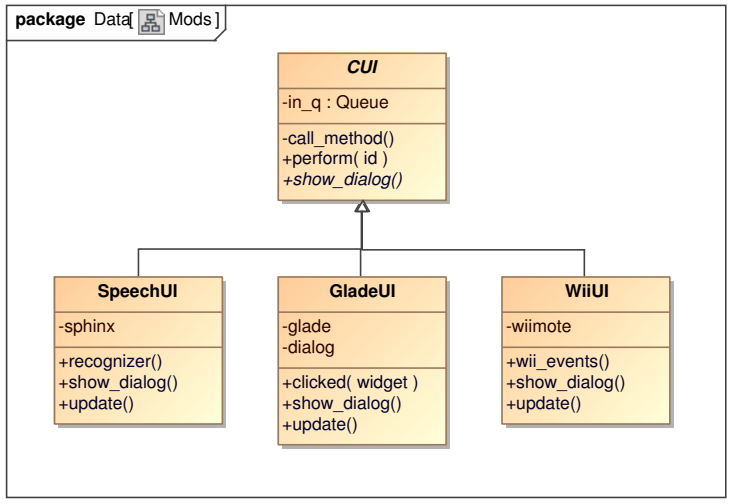


Abbildung 6.8: Ableitung der Klassen `SpeechUI`, `GladeUI` und `WiiUI` von der Klasse `CUI`.

Grafische Benutzerschnittstelle

Die für den MP3-Player erstellte GUI verwendet Glade und damit GTK2 [199, 198]. Das mArch-Framework ermöglicht, dass statt dessen ein beliebiges anderes Toolkit verwendet werden kann. Die Glade-Komponente ist erweiterbar, so dass z. B. auch die Mechanismen für Internationalisierung noch genutzt werden können [116].

In Abschnitt 4.5.3 wurde bereits der Glade User Interface Designer vorgestellt, mit dem grafische Oberflächen modelliert werden können. In Abbildung 6.9 ist eine GUI für den MP3-Player zu sehen, die mit dem User Interface Designer erstellt wurde.

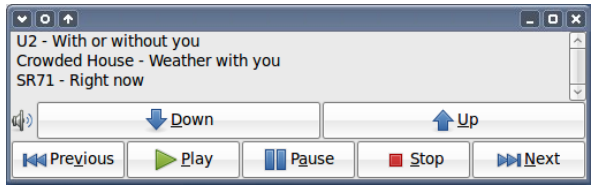


Abbildung 6.9: GUI des MP3-Players.

Die Klasse `GladeUI` in Abbildung 6.8 lädt die Beschreibung der Oberfläche in ihrem Konstruktor. Sie besitzt eine Zustandsvariable, in der sie eine Referenz auf das aktuelle Dialogfenster speichert. Sobald sie vom Mapper eine Nachricht erhält, aktualisiert sie die Zustandsvariable und das aktuelle Dialogfenster.

Die Benutzereingaben werden durch Event-Handler verarbeitet. Alle Benutzereingaben werden durch einen einzigen Handler, `clicked`, behandelt. Dieser Handler wertet aus, um was für eine Eingabe es sich handelt, und bestimmt, welche Nachricht an den Mapper übertragen werden muss.

Sprachbedienschnittstelle

Die Sprachbedienschnittstelle ermöglicht die Ein- und Ausgabe von Sprache zur Bedienung von Anwendungen. Die Klasse `SpeechUI` in Abbildung 6.8 verfügt über eine zusätzliche Methode `recognizer`, über welche eine kontinuierliche Spracheingabe erfolgt.

Zur Eingabe von Sprache wird der Sphinx Spracherkenner der CMU Sphinx Group von der Carnegie Mellon Universität eingesetzt [70]. Die Eingaben sind Kommando-basiert, d. h. der Erkenner liefert Schlüsselworte, sobald ein Wort mit einer ausreichenden Wahrscheinlichkeit erkannt wurde. Tabelle 6.3 führt die Kommandos und ihre zugehörigen Phoneme auf. Für manche der Kommandos

Kommando	Phoneme
NEXT	N EH K S
NEXT(1)	N EH K S T
PAUSE	P AO Z
PLAY	P L EY
PREVIOUS	P R IY V IY AH S
SOFTER	S AO F T ER
SOFTER(1)	S AA F T ER
STOP	S T AA P

Tabelle 6.3: Sprachkommandos für den MP3-Player und ihre Phoneme.

lassen sich mehrere Aussprache-Varianten angegeben. In der Tabelle sind dies „NEXT“ und „SOFTER“.

Zur Ausgabe von Sprache kommt das Text-to-Speech System eSpeak zum Einsatz [48]. Es unterstützt verschiedene Sprachen und auch Aussprachevarianten wie unterschiedliche Stimmlagen.

In Konfigurationsdateien für die Sprachbedienschnittstelle werden die Kommandos für die Eingabe und Sätze für die Ausgabe abgelegt. Diese Dateien werden von der Klasse `SpeechUI` beziehungsweise dem Sphinx-Erkenner geladen, so dass während der Laufzeit einer Anwendung die verschiedenen Kommandos und Ausgaben eingesetzt werden können.

Tangible User Interface (TUI)

Wie in Abschnitt 2.2.5 in Kapitel 2 „Grundlagen“ bereits beschrieben wurde, gibt es eine große Anzahl an verschiedenen Interaktionsarten bei Tangible User Interfaces. Es existiert noch keine einheitliche Notation und jedes TUI wird einzeln auf eine Anwendung zugeschnitten.

In dieser Arbeit wird die Wii Remote der Firma Nintendo als einfaches TUI eingesetzt [128]. Die Wii Remote, auch Wiimote abgekürzt, ist ein haptisches Gerät für die Wii Spielekonsole. Sie verfügt über Beschleunigungssensoren, eine Infrarotkamera und mehrere Buttons zur Eingabe. Für Ausgaben können vier LEDs, ein Tonmodul und ein Vibrationsmechanismus eingesetzt werden. Abbildung 6.10 zeigt die Wiimote mit ihren räumlichen Eingabemöglichkeiten.

In der Realisierung implementiert die Klasse `WiiUI` die Schnittstelle zu einer Wiimote (siehe Abbildung 6.8). Die Methode `wii_events` bearbeitet alle Eingabeereignisse, die von einer Wiimote kommen. Für den MP3-Player werden mehrere Buttons und zwei Gesten für die Interaktion genutzt. In Tabelle 6.4 sind alle Aktionen, die mit der Wiimote ausgeführt werden können, aufgeführt.

Aktion mit Wiimote	Kommando
Button B	Play
Button A	Pause
Wippe Rechts	Nächstes Lied
Wippe Links	Vorheriges Lied
Button Plus	Lauter
Button Minus	Leiser
Nicken nach oben	Vorheriges Lied
Nicken nach unten	Nächstes Lied

Tabelle 6.4: Aktionen mit der Wiimote zur Steuerung des MP3-Players.

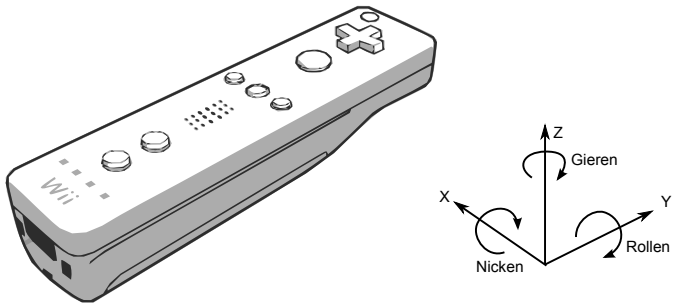


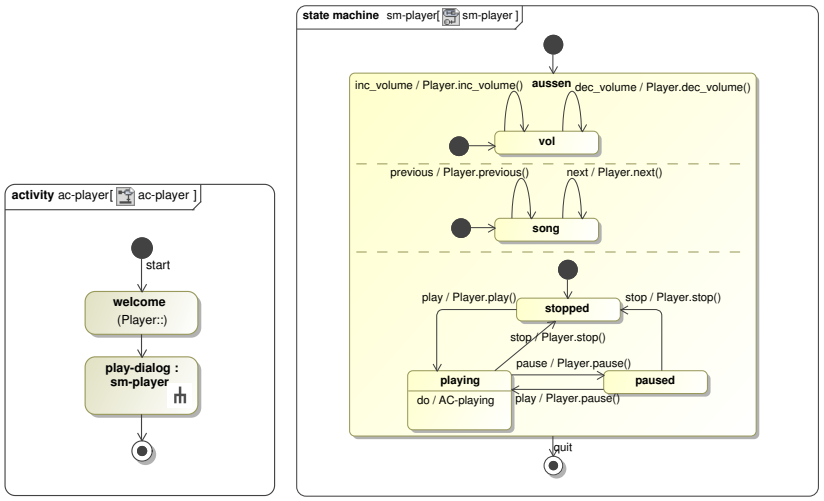
Abbildung 6.10: Die Wiimote mit ihren räumlichen Eingabemöglichkeiten.

Zusammenspiel der Komponenten

Beim Start einer Anwendung wird nach der Initialisierung der verschiedenen Komponenten des mArch-Frameworks die Kontrolle über die Interaktionssteuerung der Dialogsteuerung übergeben. Diese interpretiert die geladenen UML-Diagramme und betritt die erste Aktion eines Aktivitätsdiagramms. Die Dialogsteuerung sendet anschließend eine Nachricht an den Mapper, der die Nachricht an die verschiedenen beteiligten Modalitäten verteilt. In Abbildung 6.11a wird zuerst die Aktion `welcome` betreten. Die GUI des MP3-Players zeigt zur Begrüßung des Benutzers einen kurzen Begrüßungsdialog an. Durch die Sprachschnittstelle wird eine Begrüßung ausgegeben und die Wiimote läßt ihre LEDs aufleuchten.

Nach Beendigung der Begrüßung wird die Aktion `sm-player` betreten. Diese Aktion führt zur Anzeige der GUI in Abbildung 6.9. Sobald der Benutzer eine Schaltfläche der GUI betätigt, z. B. *Wiedergabe*, empfängt die Komponente der GUI vom Toolkit ein entsprechendes Event. Der Name des Events wird von der

GUI als Nachricht an den Mapper gesendet. Da mit dem MP3-Player fast immer eine Musikausgabe erfolgt, wird eine Sprachausgabe während der Bedienung vermieden. Auch über die Wiimote werden keine Ausgaben vorgenommen. Eingaben über Sprachbefehle oder Aktionen mit der Wiimote werden wie bei der GUI in Nachrichten für den Mapper gewandelt. Der Mapper kann mehrere Eingaben bei Bedarf fusionieren und reicht das Ergebnis als Trigger an die Dialogsteuerung weiter. Die Dialogsteuerung verarbeitet den Trigger und betritt gegebenenfalls neue Zustände oder führt Aktionen aus. Die Dialogsteuerung informiert den Mapper über Veränderungen, damit dieser ein Update der Benutzerschnittstellen bewirken kann. Durch die damit verbundene Systemantwort schließt sich der Kreis der Interaktion zwischen Dialogsteuerung und Benutzer.



(a) Aktivitätsdiagramm MP3-Player (b) Zustandsdiagramm MP3-Player

Abbildung 6.11: Links das Aktivitätsdiagramm des MP3-Players, rechts das zugehörige Zustandsdiagramm.

6.2.6 Fusion und Fission

In Abschnitt 5.3 wurden bereits verschiedene Ansätze für Fusion und Fission vorgestellt. In der Realisierung sind zwar weit komplexere Verfahren zur automatischen Generierung von Benutzerschnittstellen möglich, da diese jedoch außerhalb des Fokus dieser Arbeit liegen, wurde ein Ansatz zur direkten Anbindung der Modalitäten gewählt.

Die Fusion der Eingaben lässt sich durch eine Vielzahl von Verfahren bewältigen. In dieser Arbeit wurde die Mapper-Komponente so realisiert, dass der Fusionsmechanismus einfach austauschbar ist. So können in späteren Arbeiten die Wirksamkeiten verschiedener Mechanismen miteinander verglichen werden.

6.3 Anwendung des Frameworks

Es ist ein Ziel der Arbeit, eine einfache Verwendung der entwickelten Frameworks zu ermöglichen. Die Klassen wurden in diesem Kapitel bereits weitgehend vorgestellt. Das erstellte Framework verbirgt die meisten seiner Klassen vor dem Anwender. In Abbildung 6.12 sind die wichtigsten Klassen des Frameworks zu sehen, mit denen Entwickler in Berührung kommen.

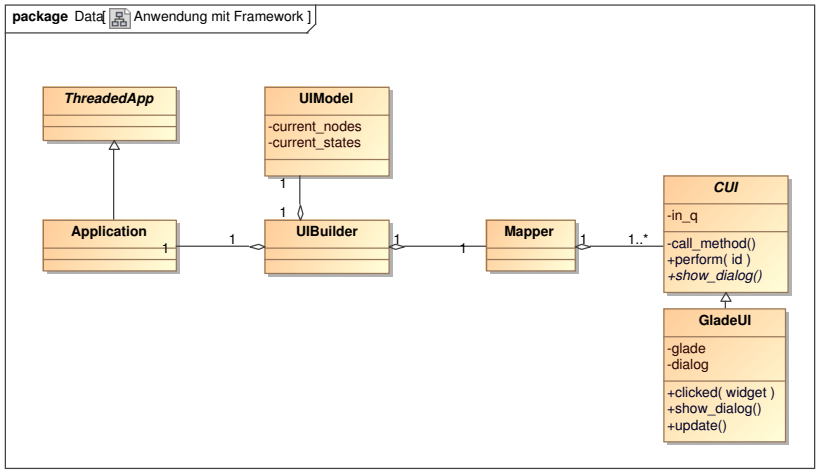


Abbildung 6.12: Die wichtigsten Klassen des Frameworks.

Im Folgenden wird nun beschrieben, wie Entwickler das Framework einsetzen können, um Anwendungen zu schreiben und neue Modalitäten in das Framework zu integrieren.

6.3.1 Regulärer Einsatz

Im normalen Einsatz des Frameworks werden Anwendungen modelliert. Der Getränkeautomat dient im Folgenden als Beispielanwendung.

Zuerst müssen Diagramme entwickelt werden, welche die Funktionalität der entsprechenden Anwendung und das Verhalten der Benutzerschnittstelle festlegen. Beim Getränkeautomat sind dies ein Klassendiagramm, ein Aktivitätsdiagramm und zwei Zustandsdiagramme (vgl. die Abbildungen 4.2a, 4.4b, 4.6 und 4.7).

Im zweiten Schritt sind Beschreibungen für die Modalitäten zu erstellen. Beim Getränkeautomat wird eine GUI mit dem Glade User Interface Designer erstellt (siehe Abschnitt 4.5.3). Für weitere Modalitäten werden ebenfalls Beschreibungen der Benutzerschnittstellen erstellt, z. B. mit VoiceXML oder einem anderen Toolkit für eine Sprachschnittstelle und mit direkter Ausprogrammierung eine Benutzerschnittstelle für TUIs.

```
1 app = Drinksvendor.new()
2 ui = UIBuilder.new(app, "drinksvendor.mdzip")
3 ui.start()
```

Listing 6.1: Starten des Getränkeautomats durch Verwendung des UIBuilders in der Sprache Ruby.

Parallel zur Erstellung der Benutzerschnittstellen für die verschiedenen Modalitäten kann die Anwendung selbst entwickelt werden. Für die Nutzung des Frameworks sind nur wenige Zeilen zusätzlicher Code nötig. Aus Listing 6.1 wird ersichtlich, wie der Getränkeautomat initialisiert und seine Benutzerschnittstelle gestartet wird. In Zeile 2 wird dem `UIBuilder` eine Referenz auf die Anwendung und der Pfad zur XMI-Datei mit den UML-Diagrammen übergeben. Danach muss nur noch in Zeile 3 die Steuerung der Anwendung an die Dialogkomponente übergeben werden.

Wie bereits in Abschnitt 6.1.2 beschrieben initialisiert und verbindet die Komponente durch Einsatz des *Builder*-Patterns alle beteiligten anderen Komponenten. Dies sind die Dialogsteuerung (`UIModel`), der Mapper sowie die verschiedenen Modalitäten.

6.3.2 Integration neuer Modalitäten

Im Framework werden alle Klassen, die eine Modalität beschreiben, von der Klasse `CUI` abgeleitet. Diese Klasse beschreibt ein *Concrete User Interface*, d. h. eine Ausprägung eines Präsentationsmodells auf Toolkitebene. In Abbildung 6.13 ist die Klasse `CUI` zu sehen. Sie besitzt ein Attribut `in_q`, welches eine Nachrichtenqueue ist, über die sie vom Mapper informiert wird. Dazu dient auch die Methode `perform(id). call_method` ist eine Hilfmethode. Sie wird eingesetzt, um Variablen abzufragen die ausgegeben werden sollen. Die Integration neuer Modalitäten in das Framework wird am Beispiel der bereits verwendeten GUI beschrieben.

Damit eine Modalität von der Anwendung entkoppelt werden kann, müssen Informationsrepräsentation und Interaktionslogik getrennt werden. Für die Realisierung wurden Glade und damit das GTK Toolkit ausgewählt, da sie eine einfache Anbindung von Beschreibungen der Benutzerschnittstelle ermöglichen. Die Dialoge der GUI werden mit dem in Abschnitt 4.5.3 beschriebenen User Interface Designer erstellt, der sie in der Glade-XML Beschreibung ablegt. Zur Laufzeit wird diese Beschreibung eingelesen und durch eine eingebundene Bibliothek angezeigt.

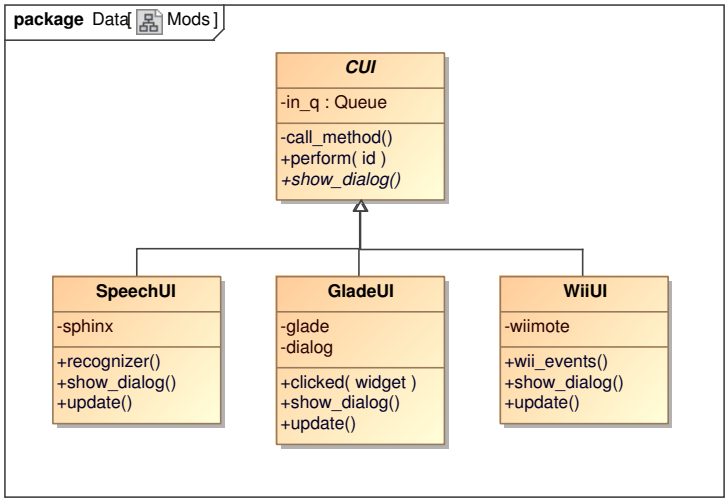


Abbildung 6.13: Ableitung der Klasse `GladeUI` von der Klasse `CUI`.

In der unteren Hälfte von Abbildung 6.13 sind die Klassen `SpeechUI`, `GladeUI` und `WiiUI` abgebildet. Um eine neue Modalität zu erstellen, wird eine neue Klasse von der Klasse `CUI` abgeleitet. Je nach Erfordernissen wird sie um Attribute angereichert. Die GUI-Klasse besitzt z. B. das Attribut `glade` für die interne Repräsentation der Oberflächenbeschreibung und das Attribut `dialog`, um zu speichern, welches Dialogfenster gerade angezeigt wird. Die Klasse für die Sprachschnittstelle besitzt ein Attribut für den Spracherkenner und die Klasse `WiiUI` ein Attribut zum Zugriff auf die Wiimote.

Eine Klasse für eine neue Modalität muss weiter um Methoden zur Bearbeitung der Eingabe- und Ausgabeereignisse verfügen. Bei der Klasse `GladeUI` werden die Eingaben des Benutzers durch die Methode `clicked` abgefangen, die als Event-Handler registriert wird. Die Eingaben werden als Nachrichten an den Mapper gesendet. Durch die Methode `show_dialog` werden die Nachrichten

des Mappers ausgewertet und die entsprechenden Dialoge angezeigt. Die Hilfsmethode `update` dient dazu, Aktualisierungen an der Oberfläche vorzunehmen. Die Sprachschnittstelle und die Klasse `WiiUI` implementieren beide die abstrakte Methode `show_dialog` und verfügen auch beide über Hilfsmethoden, um ihre internen Zustände und die Ausgabe zu aktualisieren.

Der Aufwand um neue Modalitäten in das Framework zu integrieren ist gering. Es ist zu beachten, dass keine spezifische Anwendungslogik in eine Modalitätskomponente integriert wird. Abgeleitete Klassen müssen die Nachrichten des Mappers verarbeiten können und Mechanismen implementieren, um Benutzereingaben an den Mapper senden zu können.

6.4 Fazit

In diesem Kapitel wurde das Software-Framework vorgestellt, welches die in Kapitel 5 „Architekturen für multimodale Benutzerschnittstellen“ entwickelte Architektur implementiert.

Die Fallbeispiele verdeutlichen, dass sowohl die Architektur als auch das Software-Framework funktionieren. Sie zeigen, dass die Integration des Domänenmodells auf einfache Weise möglich ist. Wie bereits im Kapitel 4 „Modellieren von Interaktionsanwendungen mit UML“ beschrieben, lassen sich die UML-Diagramme zur Interaktionsbeschreibung hierarchisch strukturieren. Dieses Kapitel erläutert mit dem neu entwickleten Kaktusstack ein Verfahren, mit dem diese Hierarchisierung umgesetzt wird.

Das Präsentationsmodell wird durch drei Modalitäten, eine grafische Benutzerschnittstelle, eine Sprachbedienschnittstelle und ein Tangible User Interface, realisiert. Multimodale Anwendungen sind durch das Framework mit geringem Aufwand möglich. Die drei Modalitäten nutzen die Fission der Mapper-Komponente. Im Rahmen der Arbeit war nur eine einfache Fission möglich, Verfahren, die zur Laufzeit eine Fission vornehmen oder vor der Ausführung einer Anwendung Benutzerschnittstellen generieren, wurden nicht aufgenommen. Diese Verfahren bilden ein Forschungsthema, das gesondert untersucht werden sollte.

Im Framework ist ein austauschbarer Fusionsmechanismus enthalten, über den auch komplexe Fusionsverfahren genutzt werden können. In der Arbeit wurde ein sehr einfacher Mechanismus implementiert, für einen zukünftigen Einsatz sind komplexere Verfahren zu untersuchen und zu implementieren.

Erweiterungen um neue Modalitäten sind auf einfache Art möglich. In der Arbeit wurden bereits drei Modalitäten erstellt, wünschenswert für Untersuchungen zur Nutzung von Multimodalität wären weitere Benutzerschnittstellen. Dies können je nach Anwendung z. B. weitere GUIs für unterschiedliche Displaygrößen oder alternative TUIs sein.

7 Fazit und Ausblick

> The purpose of computing is insight, not numbers.
>
> *Richard W. Hamming, 1962*

Ziel dieser Arbeit war es, geeignete abstrakte Modelle für die Interaktion zwischen Benutzern und Anwendungen zu finden, welche die Verknüpfungen zwischen den eingesetzten Modellen minimieren. Durch die Modelle sollte eine abstrakte Beschreibung der Interaktion eines Benutzers mit einem System dargestellt werden. Schließlich sollte eine abstrakte Interaktionsbeschreibung multimodale Benutzerschnittstellen ermöglichen, bei denen alle Modalitäten über die gleichen Interaktionsabläufe verfügen.

7.1 Beiträge

In den Kapiteln dieser Arbeit wurden eine Reihe von Aufgaben bewältigt. Im Folgenden werden die Kapitel zusammengefasst, ihre wissenschaftlichen Beiträge aufgezeigt, hervorgehoben und bewertet.

Kapitel 1 „Einleitung“ motivierte die Notwendigkeit der Arbeit. Die Problemstellung und das Ziel der Arbeit wurden vorgestellt.

In Kapitel 2 „Grundlagen“ wurden die wichtigsten Begriffe definiert. Nach Einführung des Interaktionsbegriffs folgte eine ausführliche Vorstellung der Begriffe der Modalität und Multimodalität. Die Vielfalt an Arten der Informationsrepräsentation führt zu einer großen Menge an Modalitäten. Zum Ende des Kapitels wurden Grundbegriffe der modellbasierten Softwareentwicklung und Fallbeispiele für die Arbeit eingeführt.

In Kapitel 3 „Klassifikation bestehender Ansätze“ wurde das Mapping Problem detailliert vorgestellt. Das Mapping Problem beschreibt die Schwierigkeit, zwischen verschiedenen Modellen Mappings herzustellen. Es gibt verschiedene Arten von Mappings, z. B. Verknüpfungen zwischen Modellen, Zusammensetzen

mehrerer Modelle zu einem neuen Modell und das Ableiten eines Modells aus anderen Modellen.

Um die bestehenden Ansätze einordnen zu können, wurde das **Models and Mappings-Klassifikationsframework** entwickelt, mit dem Notationen bewertet werden können. Bisherige Ansätze zur Klassifikation konzentrierten sich auf die Einordnung einzelner Aspekte aus dem Umfeld der modellbasierten Entwicklung, wie der Transformationsebenen im Cameleon-Framework. Das *Models and Mappings-Klassifikationsframework* betrachtet alle beteiligten Modelle und kann alle Mappings zwischen Modellen einer Notation erfassen. Durch dieses neue Klassifikationsframework wird eine Einordnung nach Modellabdeckung und eine Bewertung der vorhandenen Mappings ermöglicht.

Die wichtigsten Repräsentanten von Modellierungssprachen wurden mit Hilfe des Klassifikationsframeworks im Rest des Kapitels bewertet. Ein wichtiges Ergebnis der Bewertung ist die Wahl der *Unified Modeling Language* (UML) als die am besten geeignete Modellierungssprache für diese Arbeit. **Die Unified Modeling Language ist durch Use-Case-, Aktivitäts- und Zustandsdiagramme eine geeignete Notation für die abstrakte Beschreibung der Interaktion zwischen Benutzer und Anwendung.** Darüber hinaus bietet sie eine sehr weitgehende Unterstützung für Anwendungsmodelle durch Klassendiagramme und eine Reihe weiterer Sichten auf den gesamten Entwicklungsprozess durch ihre weiteren Diagrammarten. Ein großer Vorteil ist, dass Mappings innerhalb von UML bereits beschrieben sind und von Werkzeugen unterstützt werden.

Die UML ermöglicht es, komplexe Interaktionsanwendungen modalitätsunabhängig zu beschreiben. Dies war bei den bisherigen Ansätzen nur eingeschränkt durchführbar. In Abschnitt 4.5 über das Präsentationsmodell wird zusätzlich beschrieben, wie eine direkte Anbindung bzw. wahlweise Generierung von Benutzerschnittstellen möglich ist.

In Kapitel 4 „Modellieren von Interaktionsanwendungen mit UML“ folgte eine detaillierte Beschreibung, wie mit UML Interaktionsanwendungen modelliert werden können. Das Kapitel dient auch als **Verfahrensbeschreibung zur Modellierung von Interaktionsanwendungen mit UML**, indem es demonstriert, in welchem Fall welches Diagramm eingesetzt wird. Die UML umfasst 14 Diagrammarten, von denen vier eingesetzt werden: Klassendiagramme zur Beschreibung des Domänenmodells, Use-Case- und Aktivitätsdiagramme, um das Aufgabenmodell zu gliedern und zu beschreiben, und Zustandsdiagramme, mit denen das Verhalten innerhalb des Dialogmodells beschrieben wird.

Durch den Einsatz von UML lassen sich Domänen-, Aufgaben- und Dialogmodell auf detailliertere Weise beschreiben als bei den anderen vorgestellten Notationen. Die Verbindung der Diagramme untereinander wird durch den Einsatz von UML stark vereinfacht.

Damit die erarbeiteten Konzepte umgesetzt werden können, bedarf es einer geeigneten Software-Architektur. In Kapitel 5 „Architekturen für multimodale Benutzerschnittstellen“ wurden bestehende Ansätze beschrieben, die allerdings ein-

zeln für sich genommen Schwächen aufweisen, wenn sie für multimodale Benutzerschnittstellen eingesetzt werden sollen. Das mArch-Modell ist eine erweiterte Software-Architektur, für die bestehende Ansätze kombiniert und perfektioniert wurden. **Das mArch-Modell ermöglicht multimodale Interaktionsanwendungen mit einer zentralen Dialogsteuerung.** Die einzelnen Modalitäten lassen sich auf verschiedene Plattformen verteilen.

Zum Schluss der Arbeit wurde in Kapitel 6 „Realisierung eines Software-Frameworks“ das entwickelte **Software-Framework für ein multimodales User Interface Management System** zur Erstellung von Anwendungen vorgestellt. Es realisiert die beschriebene Architektur und ist mit geringem Aufwand um neue Modalitäten erweiterbar. Das Framework verwendet die eingeführten UML Diagramme zur Beschreibung der Interaktion und um festzustellen, welche Operationen auf Anwendungsklassen ausgeführt werden. Die Anwendung des Frameworks und die einfache Integration neuer Modalitäten wurde an den Fallbeispielen demonstriert.

7.2 Ausblick

Die vorliegende Arbeit erschließt eine Reihe von verwandten interessanten Forschungsthemen. Eine Reihe mit dieser Arbeit verwandter Forschungsarbeiten entsteht zur Zeit im Sonderforschungbereich Transregio 62, der an den Universitäten Ulm und Magdeburg eingerichtet wurde. Direkt mit Teilen der Arbeit verbunden sind Themen im Bereich von Fusion, Fission und Modalitäten. Hier existieren einige experimentelle und theoretische Fragestellungen für Interaktionsanwendungen.

Das Anwendungsframework verfügt über einen Plugin-Mechanismus für Fusionsalgorithmen. Experimentelle Untersuchungen zum Vergleich verschiedener Fusionsalgorithmen existieren bisher noch nicht. Durch die Nachvollziehbarkeit des Anwendungsframeworks ist eine Vergleichbarkeit bei der Bewertung des Verhaltens verschiedener Algorithmen gegeben.

Weitere interessante experimentelle Untersuchungen bieten sich zum Einsatz von Modalitäten an. Derzeit existieren nur wenige Arbeiten, die ermitteln, unter welchen Bedingungen eine spezielle Modalität zur Interaktion mit dem Benutzer besonders geeignet ist.

Das Anwendungsframework lässt sich um zusätzliche Modalitäten erweitern. Diese Erweiterungen können zum Testen von neuen Interaktionsmechanismen in z. B. Tangible User Interfaces sowie zum Zusammenspiel von multiplen Modalitäten genutzt werden.

Neue Gebiete erschließen sich bei der übergreifenden Nutzung von Modalitäten[1], die bisher wenig untersucht ist. Für sie ist eine semantische Beschreibung

[1] Transmodalität, die überlappende Nutzung von Modalitäten innerhalb eines Tasks. Vergl. Abschnitt 2.2.3.

der Interaktionselemente und ihrer Verknüpfungen ein neues Themengebiet. Durch eine solche Beschreibung lassen sich Zuordnungen über die Grenzen von Modalitäten hinweg durch die semantischen Zusammenhänge inferieren.

Ein weiteres, erst teilweise erschlossenes Themenfeld ist die Untersuchung von Verfahren zur automatisierten Fission zur Laufzeit. In dieser Arbeit wurden Beschreibungen von Benutzerschnittstellen erzeugt und durch das Dialogsystem angesteuert. Es ist jedoch auch möglich, die Benutzerschnittstellen zu generieren. In diesem Fall kann eine Fission in mehreren Stufen von der abstrakten Interaktionsbeschreibung zu Benutzerschnittstellen vorgenommen werden.

Ausgehend von der vorliegenden Arbeit wurde bereits eine Diplomarbeit zur Generierung prototypischer GUIs betreut, die viel versprechende Ergebnisse als Vorstudie lieferte [42]. Eine weitere Arbeit untersucht derzeit regelbasierte Ansätze zur automatischen Generierung [91]. In der vor kurzer Zeit begonnenen Dissertation von Mark Poguntke wird ebenfalls auf das Thema der automatischen Generierung von grafischen Benutzerschnittstellen und der Ableitung von Transformationsregeln für abstrakte Interaktionsbeschreibungssprachen eingegangen [156]. Das Thema wird dabei im Kontext von Benutzerschnittstellen in Kraftfahrzeugen behandelt. Im Kontext des Sonderforschungsbereichs Transregio 62 stehen die begonnenen Arbeiten von Frank Honold und Felix Schüssel, die sich mit automatisierter Fission im Zusammenhang mit Kontextmodellen und der adaptiven Fusion von Eingaben befassen [68, 172].

Zum Gebiet der Kontextmodelle wie Benutzereigenschaften, Gerätemodell und Umgebung ergibt sich durch diese Arbeit ein neues Betätigungsfeld. Über Kontextmodelle wirken eine Vielzahl von Kräften auf die Benutzerschnittstelle ein. Zu untersuchen ist, wie Kontextmodelle beschrieben werden können, so dass Mappings zu UML-Diagrammen auf einfache Weise möglich werden. Wünschenswert sind auch Ableitungsregelsätze, die beschreiben, in welchen Kontexten Benutzerschnittstellen auf welche Weisen angepasst werden müssen. Solche Regelsätze können dann zur Generierung von Benutzerschnittstellen oder zur Fission während der Laufzeit einer Anwendung eingesetzt werden.

A UML Begriffe Englisch – Deutsch

Die UML findet weltweit Einsatz und es wird auch verstärkt auf Deutsch über sie geschrieben. In dieser Arbeit wurden weitgehend die deutschen Übersetzungen der Begriffe verwendet. Die folgende Übersicht fasst die verwendeten Begriffe zusammen und stützt sich auf die Definition von Jeckle [83].

action Aktion

activity Aktivität

activity diagram Aktivitätsdiagramm

(activity) edge Kante

activity final node Endknoten

activity node Aktivitätsknoten oder Schritt

aggregation Aggregation, Teile/Ganzes-Beziehung

association Assoziation (ungerichtet)

behavior Verhalten

class diagram Klassendiagramm

concurrent nebenläufig

constraint Einschränkung

control edge Ablaufkante

decision Entscheidung

dependency Abhängigkeit

entry action Eintrittsaktion

event Ereignis

exit action Austrittsaktion

final state Endzustand

flow final node Ablaufende

generalization Generalisierung

guard Bedingung

initial node Startknoten

initial state Anfangszustand

interaction Interaktion

interaction diagram Interaktionsdiagramm

interruptible activity region Unterbrechbarer Aktivitätsbereich

lifeline Lebenslinie

message Nachricht, Botschaft

method Methode

node Knoten

object edge Objektkante

object node Objektknoten

pattern Muster

protocol state machine Protokoll-Zustandsautomat

state Zustand

statechart diagram Zustandsdiagramm

state machine Zustandsautomat

substate Teilzustand

superclass Oberklasse

transition Transition, Übergang

use case Use-Case, Anwendungsfall

use case diagram Use-Case-Diagramm, Anwendungsfalldiagramm

Literaturverzeichnis

[1] ActiveCharts. http://activecharts.informatik.uni-ulm.de/, 2007.

[2] ABOWD, Gregory; BOWEN, Jonathan P.; DIXM, Alan; HARRISON, Michael; TOOK, Roger: User Interface Languages: A Survey of Existing Methods. Technical report, Oxford University Computing Laboratory, October 1989.

[3] ABOWD, Gregory D.: Agents: Communicating interactive processes. In: *INTERACT '90: Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, 143–148. North-Holland. ISBN 0-444-88817-9, 1990.

[4] AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D.: *Compilers. Principles, Techniques and Tools*. Addison-Wesley Longman, 1985.

[5] ALEXANDER, Heather: *Formally-based Tools and Techniques for Human-Computer Dialogues*. Ellis Horwood, 1987.

[6] ANNETT, John; DUNCAN, Keith D.: Task Analysis and training design. *Occupational Psychology*, 41 211–221, 1967.

[7] ANONYMOUS: Mouse Gestures Redox. https://addons.mozilla.org/en-US/firefox/addon/39, 2008.

[8] ASSOCIATION FOR COMPUTING MACHINERY: The 1998 ACM Computing Classification System. http://www.acm.org/about/class/ccs98-html, 1988.

[9] ASSOCIATION FOR COMPUTING MACHINERY: Curricula for Human Computer Interaction. http://www.acm.org/sigchi/cdg/cdg2.html, 1992.

[10] AXELSSON, Jonny; CROSS, Chris; FERRANS, Jim; MCCOBB, Gerald; RAMAN, T. V.; WILSON, Les: XHTML+Voice Profile 1.2. http://www.voicexml.org/specs/multimodal/x+v/12/, March 2004.

[11] BARCLAY, Peter J.; KENNEDY, Jessie: Teallach's presentation model. In: *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, 151–154. ACM, New York, NY, USA. ISBN 1-58113-252-2, 2000.

[12] BASS, Len; FANEUF, Ross; MAYER, Niel; PELLEGRINO, Bob; REED, Scott; SEACORD, Robert; SHEPPARD, Sylvia; SZCZUR, Martha R.: A metamodel for the runtime architecture of an interactive system: The UIMS tool developers workshop. *ACM SIGCHI Bulletin*, 24 (1) 32–37, January 1992.

[13] DEN BERGH, Jan Van; LUYTEN, Kris; CONINX, Karin: High-Level Modeling of Multi-user Interactive Applications. In: CONINX, Karin; LUYTEN, Kris; SCHNEIDER, Kevin A. (Eds.), *TAMODIA '06: Proceedings of the 5th annual conference on Task modelsand diagrams*, volume LNCS 4385, 153–168. Springer, October 2006.

[14] BERNE, Eric: *Transactional Analysis in Psychotherapy*. Grove Press, New York, 1961.

[15] BERNSEN, Niels Ole: A Reference Model for Output Information in Intelligent Multimedia Presentation Systems. In: *ECAI 96 workshop: Towards a Standard Reference Model for Intelligent Multimedia Systems*, 1996.

[16] BERNSEN, Niels Ole; VERJANS, Steven: *From Task Domain to Human-Computer Interface - Exploring an Information Mapping Methodology*, 28. AAAI Press, Menlo Park. `http://www.nis.sdu.dk/~nob/publications/IMAP-29.3.pdf`, 1997.

[17] BERTI, Silvia; MORI, Giulio; PATERNÒ, Fabio; SANTORO, Carmen: TERESA: A transformation-based environment for designing multi-device interactive applications. In: *Proceedings of the 9th international conference on Intelligent user interface*, 352–353. ACM, New York, NY, USA. ISBN 1-58113-815-6, 2004.

[18] BMW GROUP: BMW 5 Series Sedan: Head-Up Display. `http://www.bmw.com/com/en/newvehicles/5series/sedan/2007/allfacts/ergonomics/hud.html`, 2007.

[19] BRAUN, Thomas; RÖHLER, Annette; WEBER, Florian: *Kurzlehrbuch Physiologie*. Elsevier, Urban und Fischer, München, 1. auflage edition, 2006.

[20] BURROWS, Daniel: aptitude. `http://packages.debian.org/aptitude`.

[21] BÖRGER, Egon; STÄRK, Robert: *Abstract State Machines*. Springer, 2003.

[22] CAFFIAU, Sybille; GIRARD, Patrick; SCAPIN, Dominique; ; GUITTET, Laurent: Generating Interactive Applications from Task Models: A Hard Challenge. In: WINCKLER, Marco; JOHNSON, Hilary; PALANQUE, Philippe (Eds.), *TAMODIA '07: Proceedings of the 6th annual conference on Task models and diagrams*, volume LNCS 4849, 267–272. Springer, November 2007.

[23] CALVARY, Gaëlle; COUTAZ, Joëlle; NIGAY, Laurence: From single-user architectural design to PAC*: a generic software architecture model for CSCW. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 242–249, 1997.

[24] CALVARY, Gaëlle; COUTAZ, Joëlle; THEVENIN, David: Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism. In: *Proc. HCI-IHM 2001, A. Blandford, J. Vanderdonckt, P. Gray Eds.,BCS conference series, Springer Publ*, 349–363, 2001.

[25] CALVARY, Gaëlle; COUTAZ, Joëlle; THEVENIN, David; LIMBOURG, Q.; SOUCHON, N.; BOUILLON, L.; VANDERDONCKT, Jean: Plasticity of User Interfaces : A revised reference framework. In: *First International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2002, Bucarest*, 127–134, 2002.

[26] CALVARY, Gaëlle; COUTAZ, Joëlle; THEVENIN, David; LIMBOURG, Quentin; BOUILLON, Laurent; VANDERDONCKT, Jean: A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15 289–308, 2003.

[27] CANON INC.: Canon EOS ELAN 7NE. `www.canon.com`. Eye Control available on EOS Elan 7NE only, 2009.

[28] CARD, S. K.; MORAN, T. P.; NEWELL, A.: *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates, New Jersey, 1983.

[29] CARD, Stuart K.; MACKINLAY, Jock D.; ROBERTSON, George G.: The design space of input devices. In: *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people*, 117–124, 1990.

[30] CARD, Stuart K.; MACKINLAY, Jock D.; ROBERTSON, George G.: A morphological analysis of the design space of input devices. In: *ACM Transactions on Information Systems*, 99–122. ACM, ACM Press, 1991.

[31] CARD, Stuart K.; MORAN, Thomas: User technology - from pointing to pondering. In: *Proceedings of the ACM Conference on The history of personal workstations*, 183–198. ACM, ACM Press, 1986.

[32] CARTER, Scott; HURST, Amy; MANKOFF, Jennifer; LI, Jack: Dynamically adapting GUIs to diverse input devices. In: *Assets '06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and accessibility*, 63–70. ACM, New York, NY, USA. ISBN 1-59593-290-9, 2006.

[33] CASCHERA, Maria Chiara; FERRI, Fernando; GRIFONI, Patrizia: Ambiguity detection in multimodal systems. In: *AVI '08: Proceedings of the working conference on Advanced visual interfaces*, 331–334. ACM, New York, NY, USA. ISBN 0-978-60558-141-5, 2008.

[34] CLERCKX, Tim; LUYTEN, Kris; CONINX, Karin: The mapping problem back and forth: customizing dynamic models while preserving consistency. In: *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*, 33–42. ACM Press, New York, NY, USA. ISBN 1-59593-000-0, 2004.

[35] CONSTANTINE, Larry: *The Peopleware Papers: Notes on the Human Side of Programming*. Prentice Hall Professional Technical Reference. ISBN 0130601233, 2001.

[36] CONSTANTINE, Larry L.: Canonical Abstract Prototypes for Abstract Visual and Interaction Design. In: JORGE, Joaquim; NUNES, Nuno; E CUNHA, João Falcão (Eds.), *Proceedings of DSV - IS'2003*, volume LNCS 2844, 1–15. Springer, 2003.

[37] CONSTANTINE, Larry L.; LOCKWOOD, Lucy A. D.: Usage-Centered Engineering for Web Applications. *IEEE Software*, 19 (2) 42–50, March 2002.

[38] COUTAZ, J.; NIGAY, L.; SALBER, D.: The MSM framework: A Design Space for Multi-Sensori-Motor Systems. In: *3rd International Conference EWHCI 1993*, 231–241, 1993.

[39] COUTAZ, Joëlle: PAC, an object oriented model for dialog design. In: *Human-Computer Interaction — INTERACT '87*, 431–436. Elsevier Science Publishers, 1987.

[40] COUTAZ, Joëlle; NIGAY, Laurence; SALBER, Daniel; BLANDFORD, Ann; MAY, Jon; YOUNG, Richard M.: Four easy pieces for assessing the usability of multimodal interaction: the CARE properties. In: *Human-Computer Interaction, INTERACT '95, IFIP TC13 Interantional Conference on Human-Computer Interaction*, 115–120. Chapman & Hall. ISBN 0-412-71790-5, 1995.

[41] DAN R. OLSEN, Jr.: Propositional production systems for dialog description. In: *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, 57–64. ACM Press, New York, NY, USA. ISBN 0-201-50932-6, 1990.

[42] DASKE, Lisa: *Generierung prototypischer GUIs auf Basis von UML Zustands- und Aktivitätsdiagrammen*. Master's thesis, Universität Ulm, 2009.

[43] DEUTSCHE TELEKOM AG: *T-Net Bedienungsanleitung*. Bonn, 2003.

[44] DHAWALE, Pushkar; MASOODIAN, Masood; ROGERS, Bill: Bare-hand 3D gesture input to interactive systems. In: *CHINZ '06: Proceedings of the 7th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction*, 25–32. ACM, New York, NY, USA. ISBN 1-59593-473-1, 2006.

[45] DIX, Alan; FINLAY, Janet; ABOWD, Gregory D.; BEALE, Russell: *Human-computer interaction*. Prentice Hall, 2003.

[46] DIX, Alan; FINLAY, Janet; ABOWD, Gregory D.; BEALE, Russell: *Human-computer interaction*. Pearson, 2004.

[47] DUARTE, Carlos; CARRIÇO, Luís: A conceptual framework for developing adaptive multimodal applications. In: *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, 132–139. ACM Press, New York, NY, USA. ISBN 1-59593-287-9, 2006.

[48] DUDDINGTON, Jonathan: eSpeak text to speech. `http://espeak.sourceforge.net/`, 2007.

[49] DUDENREDAKTION: *Duden 01. Die deutsche Rechtschreibung*. Bibliographisches Institut, Mannheim. ISBN 3411040149, 2006.

[50] ELSHOLZ, Jan-Patrick; HERMANN, Marc; DE MELO, Guido; WEBER, Michael: Personalizable Ambient Fixture. In: *4th IET/IEEE International Conference on Intelligent Environments*, 2008.

[51] FERGUSON, Paula M.; BRENNAN, David: *Motif Reference Manual B: for OSF/Motif Release 1.2*. Thomson Learning. ISBN 1565920384, 1993.

[52] FOLEY, J.; GIBBS, C.; KOVACEVIC, S.: A knowledge-based user interface management system. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 67–72, 1988.

[53] FOLEY, James; KIM, Won Chul; KOVACEVIC, Srdjan; MURRAY, Kevin: *UIDE—an intelligent user interface design environment*, 339–384. Addison-Wesley, New York, NY, USA. ISBN 0-201-50305-0, 1991.

[54] GAJOS, Krzysztof Z.; EVERITT, Katherine; TAN, Desney S.; CZERWINSKI, Mary; WELD, Daniel S.: Predictability and accuracy in adaptive user interfaces. In: *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, 1271–1274. ACM, New York, NY, USA. ISBN 978-1-60558-011-1, 2008.

[55] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John: *Design Patterns: Elements of Reuseable Object-Oriented Software*. Addison-Wesley, 1995.

[56] GELLERSEN, Hans-Werner: *Methodische Entwicklung flexibler interaktiver Software*. Shaker Verlag, 1996.

[57] GOODGER, Ben; HICKSON, Ian; HYATT, David; WATERSON, Chris: XML User Interface Language (XUL) 1.0. `http://www.mozilla.org/projects/xul/xul.html`, 2001.

[58] HAREL, David: Statecharts: A visual formalism for complex systems. *Sci Comput Program*, 8 (3) 231–274. ISSN 0167-6423, 1987.

[59] HAREL, David: Statecharts in the making: a personal account. *Communications of the ACM*, 52 (3) 67–75. ISSN 0001-0782, 2009.

[60] Harmonia Inc., Blacksburg, Virginia, U.S.A.: *User Interface Markup Language (UIML) Specification Version 3.0 Draft*, 2002.

[61] HARTSON, H. Rex; SIOCHI, Antonio C.; HIX, D.: The UAN: a user-oriented representation for direct manipulation interface designs. *ACM Trans Inf Syst*, 8 (3) 181–203. ISSN 1046-8188, 1990.

[62] HAUCK, E. A.; DENT, B. A.: Burroughs' B6500/B7500 stack mechanism. In: *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*, 245–251. ACM, New York, NY, USA, 1968.

[63] HILL, Ralph D.: The abstraction-link-view paradigm: using constraints to connect user interfaces to applications. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 335–342. ACM Press, 1992.

[64] HINCKLEY, Ken; CZERWINSKI, Mary; SINCLAIR, Mike: Interaction and modeling techniques for desktop two-handed input. In: *Proceedings of the 11th annual ACM symposium on User interface software and technology*, 49–58, 1998.

[65] HOARE, C. A. R.: Communicating sequential processes. *Commun ACM*, 21 (8) 666–677. ISSN 0001-0782, 1978.

[66] HOARE, C. A. R.: *Communicating Sequential Processes*. Prentice-Hall. ISBN 0-13-153271-5, 1985.

[67] HOGREFE, Dieter: *Estelle, LOTOS und SDL*. Springer Verlag Berlin, 1989.

[68] HONOLD, Frank Daniel: Informationsfission bei adaptiven multimodalen Systemen (vorläufiger Titel). Dissertation, noch unveröffentlicht.

[69] HORNECKER, Eva; BUUR, Jacob: Getting a grip on tangible interaction: a framework on physical space and social interaction. In: *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, 437–446. ACM, New York, NY, USA. ISBN 1-59593-372-7, 2006.

[70] HUGGINS-DAINES, David: CMUSphinx: The Carnegie Mellon Sphinx Project. `http://cmusphinx.sourceforge.net/`, 2007.

[71] INCITS: *Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Presentation Template*, 2005.

[72] INCITS: *Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Resource Desription*, 2005.

[73] INCITS: *Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Target Description*, 2005.

[74] INCITS: *Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: Universal Remote Console*, 2005.

[75] INCITS: *Protocol to Facilitate Operation of Information and Electronic Products through Remote and Alternative Interfaces and Intelligent Agents: User Interface Socket Description*, 2005.

[76] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: ISO/IS 8807: Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on Temporal Ordering of Observational Behaviour, 1989.

[77] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: DIN EN ISO 9241-110: Grundsätze der Dialoggestaltung, 1990.

[78] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: ISO/IEC 14772-1:1998: Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language – Part 1: Functional specification and UTF-8 encoding. `http://www.iso.ch/cate/d25508.html`, 1997.

[79] ISHII, Hiroshi: The tangible user interface and its evolution. *Communications of the ACM*, 51 (6) 32–36. ISSN 0001-0782, 2008.

[80] ISHII, Hiroshi; RATTI, Carlo; PIPER, Ben; WANG, Yao; BIDERMAN, Assaf; BEN-JOSEPH, E.: Bringing clay and sand into digital design — continuous tangible user interfaces. *BT Technology Journal*, 22 (4) 287–299, October 2004.

[81] JACOB, Robert J.K.; GIROUARD, Audrey; HIRSHFIELD, Leanne M.; HORN, Michael S.; SHAER, Orit; SOLOVEY, Erin Treacy; ZIGELBAUM, Jamie: Reality-based interaction: a framework for post-WIMP interfaces. In: *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, 201–210. ACM, New York, NY, USA. ISBN 978-1-60558-011-1, 2008.

[82] JAN VAN DEN BERGH; CONINX, Karin: Model-based design of context-sensitive interactive applications: a discussion of notations. In: *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*, 43–50. ACM Press, New York, NY, USA. ISBN 1-59593-000-0, 2004.

[83] JECKLE, Mario: UML auf gut Deutsch. `http://jeckle.de/uml.de/index.html`, 2004.

[84] JECKLE, Mario: Unified Modeling Language (UML) Tools. `http://www.jeckle.de/umltools.htm`, 2004.

[85] JECKLE, Mario; RUPP, Chris; HAHN, Jürgen; QUEINS, Stefan: *UML Glasklar*. Hanser Fachbuchverlag. ISBN 3446229523, 2005.

[86] JOHN, Bonnie E.; KIERAS, David E.: The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Trans Comput-Hum Interact*, 3 (4) 320–351. ISSN 1073-0516, 1996.

[87] KASIK, David J.: A user interface management system. *SIGGRAPH Comput Graph*, 16 (3) 99–106. ISSN 0097-8930, 1982.

[88] KECHER, Christoph: *UML 2.0. Das umfassende Handbuch*. Galileo Press. ISBN 3898427382, 2006.

[89] KERER, Clemens; KIRDA, Engin: Layout, Content and Logic Separation in Web Engineering. In: *Web Engineering: Software Engineering and Web Application Development*, volume 2016 of *LNCS*, 135–147, 2000.

[90] KLEENE, Stephen C.: Representation of events in nerve nets and finite automata. *Annals of Mathematics Studies*, 34 3–41, 1956.

[91] KLUGE, Verena: *Modellgetriebene Generierung graphischer Benutzeroberflächen*. Master's thesis, Universität Ulm, voraussichtlich 2010.

[92] KNUTH, Donald E.: backus normal form vs. Backus Naur form. *Commun ACM*, 7 (12) 735–736. ISSN 0001-0782, 1964.

[93] KOVACEVIC, Srdjan: TACTICS - A Model-Based Framework for Multimodal Interaction. In: *Proceedings of the AAAI Spring Symposium on Intelligent Multi-Media Multi-Modal Systems*, 1994.

[94] KOVACEVIC, Srdjan: UML and User Interface Modeling. In: BÉZIVIN, Jean; MULLER, Pierre-Alain (Eds.), *UML*, volume 1618 of *Lecture Notes in Computer Science*, 253–266. Springer. ISBN 3-540-66252-9, 1998.

[95] KRASNER, Glenn E.; POPE, Stephen T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1 (3) 26–49, 1988.

[96] LELAND, Nancy; SHAER, Orit; JACOB, Robert J.K.: TUIMS: Laying the Foundations for a Tangible User Interface Management System. In: *Toolkit Support for Interaction in the Physical World Workshop, Pervasive 2004*, April 2004.

[97] LIMBOURG, Quentin; VANDERDONCKT, Jean; MICHOTTE, Benjamin; BOUILLON, Laurent; LÓPEZ-JAQUERO, Víctor: UsiXML: A Language Supporting Multi-path Development of User Interfaces. In: *Engineering Human Computer Interaction and Interactive Systems*, volume 3425/2005 of *LNCS*, 200–220, 2005.

[98] LIMBOURG, Quentin; VANDERDONCKT, Jean; SOUCHON, Nathalie: The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results. In: PALANQUE, Philippe; PATERNÒ, Fabio (Eds.), *Interactive Systems Design, Specification, and Verification*, volume LNCS 1946, 227–246. Springer, 2001.

[99] LINK, Stefan; SCHUSTER, Thomas; HOYER, Philip; ABECK, Sebastian: Modellgetriebene Entwicklung grafischer Benutzerschnittstellen. *i-com*, 6 37–43, 2007.

[100] LOK, Simon; FEINER, Steven; NGAI, Gary: Evaluation of visual balance for automated layout. In: *Proceedings of the 9th international conference on Intelligent user interface*, 101–108, 2004.

[101] LÓPEZ-JAQUERO, Víctor; MONTERO, Francisco; REAL, Fernando: Designing user interface adaptation rules with T: XML. In: *IUI '09: Proceedings of the 13th international conference on Intelligent user interfaces*, 383–388. ACM, New York, NY, USA. ISBN 978-1-60558-168-2, 2009.

[102] MARTIN, James; ODELL, James J.: *Object-Oriented Methods*. Prentice Hall PTR, Upper Saddle River, NJ, USA. ISBN 0136308562, 1994.

[103] MCGLASHAN, Scott; BURNETT, Daniel C.; CARTER, Jerry; DANIELSEN, Peter; FERRANS, Jim; HUNT, Andrew; LUCAS, Bruce; PORTER, Brad; REHOR, Ken; TRYPHONAS, Steph: Voice Extensible Markup Language (VoiceXML) Version 2.0. `http://www.w3.org/TR/voicexml20/`, 2004.

[104] MCTEAR, Michael F.: Spoken dialogue technology: enabling the conversational user interface. *ACM Comput Surv*, 34 (1) 90–169. ISSN 0360-0300, 2002.

[105] MEALY, George H.: A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34 (5) 1045–1079, 1955.

[106] MEGGINSON, David; BROWNELL, David: Simple API for XML. `http://www.saxproject.org/`, April 2004.

[107] MELLOR, Stephen J.; BALCER, Marc J.; MELLOR, Stephen; BALCER, Marc: *Executable UML: A Foundation for Model Driven Architecture*. Addison-Wesley Professional. ISBN 0201748045, 2002.

[108] MEYER, Joseph (Ed.): *Meyers Großes Konversations-Lexikon. Ein Nachschlagewerk des allgemeinen Wissens*. Bibliographisches Institut, 6. auflage edition, 1902—08.

[109] MILGRAM, Paul; TAKEMURA, Haruo; UTSUMI, Akira; KISHINO, Fumio: Augmented reality: A class of displays on the reality-virtuality continuum. *SPIE*, 2351 282–292, 1994.

[110] MILLER, George A.: The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review*, 63 81–97, 1955.

[111] MIRCROSOFT: Microsoft Surface. `http://www.microsoft.com/surface/`, April 2008.

[112] MONTERO, Francisco; LÓPEZ-JAQUERO, Víctor; VANDERDONCKT, Jean; GONZÁLEZ, Pascual; LOZANO, María; LIMBOURG, Quentin: Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In: *Interactive Systems*, volume 3941/2006 of *LNCS*, 161–172, 2006.

[113] MOORE, Edward Forrest: Gedanken-experiments on Sequential Machines. *Automata Studies, Annals of Mathematical Studies*, 34 129–153. Princeton University Press, 1956.

[114] MORAN, Thomas P.: Getting into a system: External-internal task mapping analysis. In: *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 45–49. ACM, New York, NY, USA. ISBN 0-89791-121-0, 1983.

[115] MORI, Giulio; PATERNÒ, Fabio; SANTORO, Carmen: Tool support for designing nomadic applications. In: *Proceedings of the 8th international conference on Intelligent user interfaces*, 141–148, 2003.

[116] MUTOH, Masao: Ruby GetText. `http://www.yotabanana.com/hiki/ruby-gettext-dev.html`, 2006.

[117] MYERS, Brad; HUDSON, Scott E.; PAUSCH, Randy: Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, 7 (1) 3–28. ISSN 1073-0516, 2000.

[118] MYERS, Brad A.: A new model for handling input. *ACM Transactions on Information Systems*, 8 (3) 289–320, 1990.

[119] MYERS, Brad A.: User interface software tools. *ACM Transactions on Computer-Human Interaction*, 2 (1) 64–103, 1995.

[120] MYERS, Brad A.; ROSSON, Mary Beth: Survey on user interface programming. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 195–202, 1992.

[121] MYNATT, Elizabeth D.; EDWARDS, W. Keith: Mapping GUIs to auditory interfaces. In: *UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology*, 61–70. ACM Press, New York, NY, USA. ISBN 0-89791-549-6, 1992.

[122] MÜLLER, Johannes: *Handbuch der Physiologie des Menschen*. J. Hölscher Verlag, dritte verbesserte auflage edition. `http://vlp.mpiwg-berlin.mpg.de/references?id=lit17251`, 1837.

[123] NASSI, I.; SHNEIDERMAN, B.: Flowchart techniques for structured programming. *SIGPLAN Not*, 8 (8) 12–26. ISSN 0362-1340, 1973.

[124] NEWMAN, William M.: A system for interactive graphical programming. In: *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*, 47–54. ACM, New York, NY, USA, 1968.

[125] NIGAY, Laurence: *Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales*. Ph.D. thesis, Laboratoire de Génie Informatique (IMAG), Université Joseph Fourier, January 1994.

[126] NIGAY, Laurence; COUTAZ, Joëlle: A design space for multimodal systems: concurrent processing and data fusion. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 172–178, 1993.

[127] NIGAY, Laurence; COUTAZ, Joëlle: A generic platform for addressing the multimodal challenge. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, 98–105, 1995.

[128] NINTENDO: Controllers at Nintendo. `http://www.nintendo.com/wii/what/controllers`, 2009.

[129] NO MAGIC INC.: Magic Draw. `http://www.magicdraw.com/`, 2008.

[130] NÓBREGA, Leonel; NUNES, Nuno Jardim; COELHO, Helder: Mapping ConcurTaskTrees into UML 2.0. In: *DSVIS'05 - 12th International Workshop on Design, Specification and Verification of Interactive Systems*, 237–248. Springer, Berlin. ISBN 978-3-540-34145-1, 2005.

[131] NORMAN, Donald A.: *The Psychology of Everyday Things*. Basic Books, 1988.

[132] NUNES, Duarte Nuno Jardim: *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach*. Ph.D. thesis, University of Madeira, April 2001.

[133] NYLANDER, Stina: *The Ubiquitous Interactor - Mobile Services with Multiple User Interfaces*. Ph.D. thesis, Uppsala University, Sweden, November 2003.

[134] NYLANDER, Stina: Semi-automatic generation of device adapted user interfaces. In: *3rd International Conference on Pervasive Computing*, 109–114, 2005.

[135] OBJECT MANAGEMENT GROUP: Model Driven Architecture. `http://www.omg.org/mda/`.

[136] OBJECT MANAGEMENT GROUP: *Diagram Interchange - Version 1.0*, April 2006.

[137] OBJECT MANAGEMENT GROUP: *Meta Object Facility (MOF) Core Specification*, January 2006.

[138] OBJECT MANAGEMENT GROUP: *Object Constraint Language Version 2.0*, May 2006.

[139] OBJECT MANAGEMENT GROUP: *MOF 2.0/XMI Mapping - Version 2.1.1*, December 2007.

[140] OBJECT MANAGEMENT GROUP: *MOF Query / Views / Transformations*, April 2008.

[141] OBJECT MANAGEMENT GROUP: *OMG Unified Modeling Language Infrastructure - Version 2.2*, February 2009.

[142] OBJECT MANAGEMENT GROUP: *OMG Unified Modeling Language Superstructure - Version 2.2*, February 2009.

[143] OVIATT, Sharon: Ten myths of multimodal interaction. *Communications of the ACM*, 42 (11) 74–81, November 1999.

[144] OVIATT, Sharon: *Multimodal interfaces*, 286–304. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA. ISBN 0-8058-3838-4, 2003.

[145] PAGE-JONES, Meilir: *Fundamentals of object-oriented design in UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0-201-69946-X, 2000.

[146] PATERNÒ, Fabio: *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK. ISBN 1852331550, 1999.

[147] PATERNÒ, Fabio; MANCINI, Cristiano; MENICONI, Silvia: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: *INTERACT*, 362–369, 1997.

[148] PATTERSON, Aaron: XML Document Parsing Comparison. `http://github.com/tenderlove/xml_truth`.

[149] PATTERSON, Aaron; DALESSIO, Mike: Nokogiri. `http://nokogiri.rubyforge.org/nokogiri/`.

[150] PAYNE, Stephen J.; GREEN, T. R. G.: Task-action grammars: a model of the mental representation of task languages. *Hum-Comput Interact*, 2 (2) 93–133. ISSN 0737-0024, 1986.

[151] PEDERIVA, Inés; VANDERDONCKT, Jean; ESPAÑA, Sergio; PANACH, Ignacio; PASTOR, Oscar: The Beautification Process in Model-Driven Engineering of User Interfaces. In: BARANAUSKAS, Cécilia; PALANQUE, Philippe; ABASCAL, Julio; BARBOSA, Simone Diniz Junqueira (Eds.), *Human-Computer Interaction – INTERACT 2007*, volume 4662/2008 of *LNCS*, 411–425, 2008.

[152] PERLIN, Ken; MEYER, Jon: Nested User Interface Components. In: *Proceedings of the 12th annual ACM symposium on User interface software and technology*, 11–18, 1999.

[153] PETRI, Carl Adam: *Kommunikation mit Automaten*. Ph.D. thesis, Schriften des Rheinisch-Westfälischen Institutes für instrumentelle Mathematik an der Universität Bonn, 1962.

[154] PFAFF, G.; HAGEN, P. (Eds.): *Seeheim Workshop on User Interface Management Systems*. Springer Verlag, Berlin, 1985.

[155] PHANOURIOU, Constantinos: *Uiml: A device-independent user interface markup language*. Ph.D. thesis, Virginia Polytechnic Institute, Blacksburg, Virgina, United States, 2000.

[156] POGUNTKE, Mark: Modellgetriebene Entwicklung flexibler Mensch-Fahrzeug-Interaktion mit externen Infotainmentanwendungen (vorläufiger Titel). Dissertation, noch unveröffentlicht.

[157] PROTZENKO, Jonathan: *XUL*. Open Source Press. ISBN 3937514236, 2007.

[158] PUERTA, Angel; EISENSTEIN, Jacob: Towards a general computational framework for model-based interface development systems. In: *Proceedings of the 4th international conference on Intelligent user interfaces*, 171–178, 1999.

[159] PUERTA, Angel; EISENSTEIN, Jacob: XIML: A Universal Language for User Interfaces. `http://www.ximl.org/pages/docs.asp`, 2001.

[160] PUERTA, Angel; EISENSTEIN, Jacob: XIML: a common representation for interaction data. In: *Proceedings of the 7th international conference on Intelligent user interfaces*, 214–215, 2002.

[161] RAISTRICK, Chris; FRANCIS, Paul; WRIGHT, John: *Model Driven Architecture with Executable UML*. Cambridge University Press. ISBN 0521537711, 2004.

[162] REENSKAUG, Trygve: Models - Views - Controllers. Technical report, Xerox Palo Alto Research Laboratory (PARC), 1979.

[163] REISIG, Wolfgang: *Abstract State Machines for the Classroom*, 15–46. Monographs in Theoretical Computer Science. Springer, 2008.

[164] REISNER, P.: Formal Grammar and Human Factors Design of an Interactive Graphics System. *IEEE Trans Softw Eng*, 7 (2) 229–240. ISSN 0098-5589, 1981.

[165] REKIMOTO, Jun: Organic interaction technologies: from stone to skin. *Commun ACM*, 51 (6) 38–44. ISSN 0001-0782, 2008.

[166] RICHTER, Kai: *Methoden zur Unterstützung bei der Entwicklung plattformübergreifender Benutzerschnittstellen*. Ph.D. thesis, Technische Universität Darmstadt, 2007.

[167] RUSSELL, Sean: REXML. `http://www.germane-software.com/software/rexml/`, March 2008.

[168] SADEK, David; DE MORI, Renato: *Dialogue Systems*, 523–561. Academic Press. ISBN 9780122090554, 1997.

[169] SANNELLA, Michael: Skyblue: a multi-way local propagation constraint solver for user interface construction. In: *Proceedings of the 7th annual ACM symposium on User interface software and technology*, 137–146, 1994.

[170] SARSTEDT, Stefan; GESSENHARTER, Dominik; KOHLMEYER, Jens; RASCHKE, Alexander; SCHNEIDERHAN, Matthias: ActiveChartsIDE - An Integrated Software Development Environment comprising a Component for Simulating UML 2 Activity Charts. In: *The 2005 European Simulation and Modelling Conference (ESM'05)*, 66–73, 2005.

[171] SCHMIDT, Robert F.; LANG, Florian (Eds.): *Physiologie des Menschen*. Springer. ISBN 978-3-540-32908-4, 2007.

[172] SCHÜSSEL, Felix: Adaptive Fusion in modellbasierten Anwendungen (vorläufiger Titel). Dissertation, noch unveröffentlicht.

[173] SEFFAH, Ahmed; JAVAHERY, Homa: *Multiple user interfaces: cross-platform applications and context-aware interfaces*, 11–26. John Wiley and Sons Ltd, 2003.

[174] SEKI, Masatoshi: druby – distributed Ruby. `http://www2a.biglobe.ne.jp/~seki/ruby/druby.html`, 2005.

[175] SHARLIN, Ehud; WATSON, Benjamin; KITAMURA, Yoshifumi; KISHINO, Fumio; ITOH, Yuichi: On tangible user interfaces, humans and spatiality. *Personal Ubiquitous Comput*, 8 (5) 338–346. ISSN 1617-4909, 2004.

[176] SHEPHERD, A.: *Analysis and training in information technology tasks*, 15–55. Ellis Horwood, Chichester, 1989.

[177] SHNEIDERMAN, Ben: *Designing the user interface:strategies for effective human-computer-interaction*. Addison-Wesley, 1998.

[178] SHNEIDERMAN, Ben: *Leonardo's Laptop: Human Needs and the New Computing Technologies*. MIT Press, 2003.

[179] DA SILVA, Paulo Pinheiro; GRIFFITHS, Tony; PATON, Norman W.: Generating user interface code in a model based user interface development environment. In: *AVI '00: Proceedings of the working conference on Advanced visual interfaces*, 155–160. ACM, New York, NY, USA. ISBN 1-58113-252-2, 2000.

[180] SOMMERVILLE, Ian: *Software Engineering*. Pearson, 8. auflage edition, 2007.

[181] SONDERGAARD, Thomas: rubyzip. `http://rubyzip.sourceforge.net/`, 2006.

[182] STOCK, Ingo; WEBER, Michael: Authoring technical documentation using a generic document model. In: *SIGDOC '06: Proceedings of the 24th annual ACM international conference on Design of communication*, 172–179. ACM, New York, NY, USA. ISBN 1-59593-523-1, 2006.

[183] STOCK, Ingo; WEBER, Michael; STEINMEIER, Eckhard: Metadata based authoring for technical documentation. In: *SIGDOC '05: Proceedings of the 23rd annual international conference on Design of communication*, 60–67. ACM, New York, NY, USA. ISBN 1-59593-175-9, 2005.

[184] STOLZE, Markus; RIAND, Philippe; WALLACE, Mark; HEATH, Terry: Agile Development of Workflow Applications with Interpreted Task Models. In: WINCKLER, Marco; JOHNSON, Hilary; PALANQUE, Philippe (Eds.), *TAMODIA '07: Proceedings of the 6th annual conference on Task models and diagrams*, volume LNCS 4849, 2–14. Springer, November 2007.

[185] SZEKELY, Pedro A.; SUKAVIRIYA, Piyawadee Noi; CASTELLS, Pablo; MUTHUKUMARASAMY, Jeyakumar; SALCHER, Ewald: Declarative interface models for user interface construction tools: the MASTERMIND approach. In: *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*, 120–150. Chapman & Hall, Ltd., London, UK, UK. ISBN 0-412-72180-5, 1996.

[186] TANAKA, Katsumi: A robust selection system using real-time multi-modal user-agent interactions. In: *Proceedings of the 4th international conference on Intelligent user interfaces*, 105–108, 1998.

[187] THOMPSON, Ken: Short Descriptions of the QEF Tools. `http://www.qef.com/html/toolsdesc.html`, 1968.

[188] TIDWELL, Jenifer: Common Ground: A Pattern Language for Human-Computer Interface Design. `www.mit.edu/~jtidwell/interaction_patterns.html`.

[189] TOURAINE, Damien; BOURDOT, Patrick; BELLIK, Yacine; BOLOT, Laurence: A framework to manage multimodal fusion of events for advanced interactions within virtual environments. In: *Proceedings of the workshop on Virtual environments 2002*, 159–168, 2002.

[190] TREWIN, Shari; ZIMMERMANN, Gottfried; VANDERHEIDEN, Gregg: Abstract user interface representations: how well do they support universal access? In: *ACM Conference on Universal Usability 2003*, 77–84. Vancouver, British Columbia, Canada, 2003.

[191] ULLMER, Brygg; ISHII, Hiroshi: *Emerging Frameworks for Tangible User Interfaces*, 579–601. Addison-Wesley, August 2001.

[192] UNDERKOFFLER, John; ISHII, Hiroshi: Urp: a luminous-tangible workbench for urban planning and design. In: *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, 386–393. ACM, New York, NY, USA. ISBN 0-201-48559-1, 1999.

[193] USER INTERFACE SOFTWARE GROUP: Garnet. `http://www.cs.cmu.edu/~garnet/`, Carnegie Mellon University.

[194] USIXML CONSORTIUM: *USIXML 1.4.5 documentation draft*. `http://www.usixml.org/`, 2005.

[195] USIXML CONSORTIUM: *UsiXML V1.8, USer Interface eXtensible Markup Language Reference Manual*. Université catholique de Louvain (UCL) , Belgium. `http://www.usixml.org/`, February 2007.

[196] VANDERDONCKT, Jean; LIMBOURG, Quentin; MICHOTTE, Benjamin; BOUILLON, Laurent; TREVISAN, Daniela; FLORINS, Murielle: UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces. In: *Proceedings of W3C Workshop on Multimodal Interaction WMI'2004*, 1–7, 2004.

[197] VARIOUS: Tcl/Tk. `http://www.tcl.tk/`, 1991.

[198] VARIOUS: The GTK+ Project. `http://www.gtk.org/`, 1997.

[199] VARIOUS: Glade - a User Interface Designer for GTK+ and GNOME. `http://glade.gnome.org/`, December 1998.

[200] VAN DER VEER, Gerrit; VAN WELIE, Martijn: Task based groupware design: putting theory into practice. In: *DIS '00: Proceedings of the conference on Designing interactive systems*, 326–337. ACM Press, New York, NY, USA. ISBN 1-58113-219-0, 2000.

[201] VERNIER, Frederic; NIGAY, Laurence: A Framework for the Combination and Characterization of Output Modalities. In: PALANQUE, P.; PATERNÒ, F. (Eds.), *Interactive Systems - Design, Specification, and Verification: 7th International Workshop, DSV-IS 2000*, volume 1946, 35–52. Springer, 2001.

[202] WATZLAWICK, Paul; BEAVIN, Janet H.; JACKSON, Don D.: *Menschliche Kommunikation. Formen, Störungen, Paradoxien*. Huber, 2000.

[203] WEIDENBACHER, Ulrich; LAYHER, Georg; BAYERL, Pierre; NEUMANN, Heiko: Detection of Head Pose and Gaze Direction for Human-Computer Interaction. In: ANDRE, Elisabeth; DYBKJÆR, Laila; MINKER, Wolfgang; NEUMANN, Heiko; WEBER, Michael (Eds.), *Perception and Interactive Technologies*, volume LNCS 4021/2006, 9–19. Springer, 2006.

[204] WEISER, Mark: Some Computer Science Issuses in Ubiquitous computing. *Communications of the ACM*, 36 (7) 74–84, 1993.

[205] WELD, D.; ANDERSON, C.; DOMINGOS, P.; ETZIONI, O.; LAU, T.; GAJOS, K.; WOLFMAN, S.: Automatically Personalizing User Interfaces. In: *Proceedings of IJCAI-03*, 7, 2003.

[206] WIECHA, Charles; BOIES, Stephen: Generating user interfaces: principles and use of it style rules. In: *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, 21–30, 1990.

[207] WIGDOR, Daniel; FORLINES, Clifton; BAUDISCH, Patrick; BARNWELL, John; SHEN, Chia: Lucid touch: a see-through mobile device. In: *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, 269–278. ACM, New York, NY, USA. ISBN 978-1-59593-679-2, 2007.

[208] WORLD WIDE WEB CONSORTIUM: Cascading Style Sheets. `http://www.w3.org/TR/CSS1`.

[209] WORLD WIDE WEB CONSORTIUM: XML Path Language (XPath). `http://www.w3.org/TR/xpath`, 1999.

[210] WORLD WIDE WEB CONSORTIUM: W3C Multimodal Interaction Framework. `http://www.w3.org/TR/mmi-framework/`. `http://www.w3.org/TR/mmi-framework/`, May 2003.

[211] WORLD WIDE WEB CONSORTIUM: Document Object Model (DOM) Level 3 Core Specification. `http://www.w3.org/TR/DOM-Level-3-Core`, April 2004.

[212] WORLD WIDE WEB CONSORTIUM: RDF/XML Syntax Specification (Revised). `http://www.w3.org/TR/rdf-syntax-grammar/`, February 2004.

[213] WORLD WIDE WEB CONSORTIUM: Document Object Model (DOM). `http://www.w3.org/DOM/`, January 2005.

[214] WORLD WIDE WEB CONSORTIUM: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 2.0. `http://www.w3.org/TR/CCPP-struct-vocab2/`, April 2007.

[215] WORLD WIDE WEB CONSORTIUM: XForms 1.0. `http://www.w3.org/TR/2006/REC-xforms-20060314/`, 2007.

[216] ZIMMERMANN, Gottfried; VANDERHEIDEN, Gregg; GILMAN, Al: Prototype implementations for a universal remote console specification. In: *Conference on Human Factors in Computing Systems*, 510–511, 2002.

Zusammenfassung

Schlüsselwörter: Mensch-Maschine Interaktion, Aufgabenmodellierung, interaktive Systeme, Benutzerschnittstellen, modellbasierte Entwicklung, User Interface Management Systeme (UIMS), objektorientiertes Design, multimodale Benutzerschnittstellen.

Einordnung in das Computing Classification System, [8]: H.1.2 User/-Machine Systems, H.5.2 User Interfaces, D.2 Software Engineering, D.2.2 Design Tools and Techniques, D.2.11 Software Architectures, I.3.6 Interaction Techniques.

Die Erstellung multimodaler Interaktionsanwendungen ist trotz vieler Fortschritte in der Anwendungsentwicklung eine große Herausforderung. Die Ursache liegt in der Vielzahl von vorhandenen Modalitäten, die unterstützt werden sollen. Für jede Modalität muss eine komplette Benutzerschnittstelle implementiert oder adaptiert werden.

Durch geeignete, abstrakte Modelle für die Interaktion zwischen Benutzer und Anwendung und ihre Anbindung an Modalitäten lassen sich Verfahren aus der modellbasierten Software-Entwicklung auf das Problem anwenden. Ein wichtiger Aspekt ist, den Aufwand für Mappings zwischen den beteiligten Modellen gering zu halten. Benutzerschnittstellen sind durch die Trennung der Repräsentation ihrer Inhalte (Oberflächen, Sprache, etc.) von Interaktionslogik einfacher zu erstellen.

In der Arbeit werden zuerst in Kapitel 2 „Grundlagen“ die wichtigsten Begriffe definiert: Interaktion, Modalität und Multimodalität. Weiter werden Grundbegriffe der modellbasierten Softwareentwicklung und Fallbeispiele eingeführt.

Kapitel 3 „Klassifikation bestehender Ansätze“ stellt das Mapping Problem detailliert vor. Das Mapping Problem beschreibt die Schwierigkeit, zwischen verschiedenen Modellen Verknüpfungen herzustellen und beizubehalten. Um existierende Ansätze einordnen zu können, wird das *Models and Mappings-Klassifikationsframework* entwickelt, mit dem Notationen bewertet werden können. Das Klassifikationsframework betrachtet alle beteiligten Modelle und kann alle

Mappings zwischen Modellen einer Notation erfassen. Mit dem Klassifikationsframework werden die wichtigsten Repräsentanten von Modellierungssprachen bewertet. Ein wichtiges Ergebnis der Bewertung ist die Wahl der *Unified Modeling Language* (UML) als die am besten geeignete Notation für Domänen- und Interaktionsbeschreibung in dieser Arbeit.

Kapitel 4 „Modellieren von Interaktionsanwendungen mit UML“ beschreibt detailliert, wie mit UML Interaktionsanwendungen modelliert werden können. Es dient auch als Verfahrensbeschreibung zur Modellierung von Interaktionsanwendungen und zeigt, in welchem Fall welches UML-Diagramm angewandt wird. Von 14 Diagrammarten der UML werden vier eingesetzt: Klassendiagramme zur Beschreibung des Domänenmodells, Use-Case- und Aktivitätsdiagramme, um das Aufgabenmodell zu gliedern und zu beschreiben, und Zustandsdiagramme, mit denen das Verhalten innerhalb des Dialogmodells beschrieben wird. Das Kapitel zeigt auch überblicksartig, wie eine direkte Anbindung bzw. wahlweise Generierung von Benutzerschnittstellen möglich ist.

Kapitel 5 „Architekturen für multimodale Benutzerschnittstellen“ stellt eine Software-Architektur vor, welche die erarbeiteten Konzepte umsetzt. Bestehende Ansätze weisen für sich genommen Schwächen auf, wenn sie für multimodale Benutzerschnittstellen eingesetzt werden sollen. Das entwickelte mArch-Modell ist eine erweiterte Software-Architektur, für die die Vorzüge bestehender Ansätze kombiniert und perfektioniert wurden. Es ermöglicht multimodale Interaktionsanwendungen mit einer zentralen Dialogsteuerung. Die einzelnen Modalitäten lassen sich auf verschiedene Plattformen verteilen.

Zum Schluss der Arbeit beschreibt Kapitel 6 „Realisierung eines Software-Frameworks“, wie die das mArch-Modell aus Kapitel 5 als Software-Framework für ein multimodales User Interface Management System zur Erstellung von Anwendungen umgesetzt wird. Das Software-Framework realisiert die beschriebene Architektur und lässt sich mit geringem Aufwand um neue Modalitäten erweitern. Die Anwendung des Frameworks und die einfache Integration neuer Modalitäten wird an den Fallbeispielen demonstriert.

Kapitel 7 „Fazit und Ausblick“ fasst die Ergebnisse der Arbeit zusammen. Wichtige Beiträge sind:

- Das *Models and Mappings-Klassifikationsframework* zur Bewertung von Notationen für Interaktionsanwendungen, der Abdeckung von Modellen und Mappings.
- Die Feststellung der Eignung von UML für die abstrakte Interaktionsbeschreibung.
- Eine Verfahrensbeschreibung zur Modellierung von Interaktionsanwendungen mit UML.
- Das mArch-Modell als Software-Architektur für multimodale Interaktionsanwendungen.

- Ein Software-Framework, welches multimodale User Interface Management Systeme auf Basis des mArch-Modells ermöglicht.

Abschließend wird ein Ausblick auf weitere Themen gegeben, an denen auf Basis der vorliegenden Arbeit geforscht werden kann.

Der Anhang enthält eine Übersetzung der wichtigsten Begriffe aus der UML zwischen Englisch und Deutsch.

Summary

Keywords: Human computer interaction, task modelling, interactive systems user interfaces, model-based development, user interface management systems (UIMS), object-oriented design methods, multimodal interfaces.

Classification according to the Computing Classification System, [8]: H.1.2 User/Machine Systems, H.5.2 User Interfaces, D.2 Software Engineering, D.2.2 Design Tools and Techniques, D.2.11 Software Architectures, I.3.6 Interaction Techniques.

The creation of multimodal interactive applications remains a major challenge despite many advances in application development. The reason is the variety of existing modalities which have to be supported. For each modality a complete user interface needs to be implemented or adapted.

By employing suitable abstract models for the interaction between user and application and connecting them to modalities, methods from model-based software development can be applied to the problem. An important aspect is to minimize the effort spent on mappings between the participating models. User interfaces can be implemented in a simpler way, if representation of content (user interface, language, etc.) and interaction logic are separated.

In this thesis, chapter 2 introduces important concepts: interaction, modalities and multimodality. It also introduces basic concepts of model-based software development and case studies.

Chapter 3 describes the mapping problem in detail. The mapping problem consists in the difficulty of establishing and maintaining links between different models. In order to classify existing approaches, the *Models and Mappings-classification framework* is developed. Existing notations are evaluated utilizing the classification framework by analysing which models are involved and capturing all mappings between models. The framework is used to classify important representatives of modeling languages. An important result of this classification is the choice of *Unified Modeling Language* (UML) as the most suitable notation for domain and interaction modeling.

Chapter 4 describes in detail how interactive applications can be modeled with UML. It also serves as a process description and demonstrates which UML diagram is employed in which case. Of the 14 UML diagrams, four are employed: class diagrams describe the domain model, use case and activity diagrams structure and describe the task model, and state diagrams describe the behavior of the dialog model. The chapter gives an overview of how user interfaces can be connected directly or how they can be generated.

Chapter 5 introduces a software architecture which implements the presented concepts. Existing approaches still contain weaknesses if they are to be used for multimodal user interfaces. The chapter introduces the mArch-model, which is an advanced software architecture that combines the advantages of existing approaches. It facilitates multimodal interactive applications with a centralized dialog control. The various modalities can be distributed to different platforms.

Towards the end of the thesis, chapter 6 describes how the mArch-model from the previous chapter is utilized in a software framework for a multimodal user interface management system for building applications. The software framework implements the described architecture and can be extended with little effort to include new modalities. The utilization of the framework and the easy integration of new modalities is demonstrated in case studies.

Chapter 7 summarizes the results of this thesis. The most important contributions to research are:

- The *Models and Mappings classification framework* for the assessment of notations for interactive applications, the coverage of models and mappings.
- The identification of UML as a suitable language for abstract descriptions of interaction.
- A process description for modeling interactive applications with UML.
- The mArch-model as a software architecture for multimodal applications.
- A software framework which allows multimodal user interface management systems based on the mArch-model.

Finally, an outlook on further research based on this work is given.

The appendix contains a translation of the main concepts from UML between English and German.

Dank

Diese Arbeit hätte ohne die Unterstützung der Menschen um mich nicht geschrieben werden können. Besonders möchte ich meinem Betreuer Prof. Michael Weber danken. Meine Kollegen aus dem Institut für Medieninformatik hatten immer ein offenes Ohr für mich, ganz besonders danke ich Marc Hermann.

Ohne meine fleißigen Korrekturleser wäre der Text nicht so gut lesbar geworden. Vielen Dank an Felix Schüssel, Frank Honold, Josef Kovacs, Mark Poguntke und ganz besonders Tom Lange und Ingo Grüll.

Der Landesgraduierten-Stiftung des Landes Baden-Württemberg verdanke ich ein Stipendium, das mir die Forschung an meinem Thema sehr erleichtert hat.

Schließlich will ich meiner Familie für ihre Liebe und Unterstützung danken.

Bildnachweis

Die Abbildungen in der vorliegenden Arbeit wurden mit Ausnahme der hier aufgeführten Liste selbst erstellt.

Die Titelseite des Buchs zeigt den Kinkaku-ji (Goldener Pavillon-Tempel) in Kyoto. Er wurde 1397 errichtet und überlebte mehrere Kriege unversehrt. 1950 wurde er von einem buddhistischen Mönch durch Brandstiftung zerstört. Der Mönch hatte die Schönheit des Gebäudes nicht mehr ertragen können. Dank der überlieferten Pläne und Skizzen konnte ein exaktes Modell erstellt werden und 1955 wurde der Kinkaku-ji restauriert.

Abbildung 2.5b HTC Touch mit freundlicher Genehmigung der Firma HTC.

Abbildung 2.6 URP, 2.7 SandScape mit freundlicher Genehmigung des MIT Media Laboratory.

Abbildung 2.8a Kaffeeautomat mit freundlicher Genehmigung von Alex (zigshot82).

Abbildung 2.9a Mit freundlicher Genehmigung von Jannes Glas (stock.xchng).